МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

_____Савченко А.С.

«___»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТРА»

ЗА СПЕЦІАЛІЗАЦІЄЮ «ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ (ЗА ГАЛУЗЯМИ)»

Тема: «Технологія сучасних односторінкових веб-додатків Single Page Application»

Виконавець: Деяк Андрій Юрійович

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер: _____ Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

Спеціальність 122 «Комп'ютерні науки»

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Савченко А.С.
«_____»_____ 2020 р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Деяка Андрія Юрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Технологія сучасних односторінкових веб-додатків Single Page Application» затверджена наказом ректора №1891/ст. від 02.10.2020р..

2. Термін виконання роботи: з 05.10.2020р. по 31.12.2020р.

3. Вихідні дані до роботи: мінімальний об'єм оперативної пам'яті (RAM) – 1 GB; будь-який установлений сучасний браузер.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, засоби і методи створення сучасних односторінкових веб-додатків Single Page Application, концепція односторінкового веб-додатку, дослідження типових особливостей архітектури веб-сайтів, дослідження особливостей розробки веб-додатку з використанням Angular 10, створення односторінкового веб-додатку з використанням Angular 10, висновок.

5. Перелік обов'язкового ілюстративного матеріалу: сервіси Google, сервіс Gmail, Сервіс Google Maps, сервіс Microsoft Excel Online, сервіс Microsoft OneDrive, сервіс Microsoft Word Online, сервіс Microsoft PowePoint Online, базова структура проекту, запуск проекту, запущений проект в браузері, веб-дизайн панелі управління, модальне вікно компонента MarketingDashboardPopoverComponent, індикатор загрузки компонент

LoaderComponent, індикатор відсутності результатів компонента NoResultsFoundComponent, життєві цикли Angular компонентів, графіки з налаштованим фільтром, графік нових товарів «New ins» з періодом часу за останній тиждень, таблиця структури цін «Price structure», статистичний блок даних, статистичний блок без вибраног фільтру.

6. Календарний план-графік

№ п/п	***Завдання***	***Термін виконання***	***Підпис керівника***
1.	Проаналізувати літературу та джерела за темою дипломної роботи	05.10.20 – 10.10.20р.	
2.	Розроблення та затвердження плану дипломної роботи	11.10.20 – 15.10.20р.	
3.	Провести консультації з науковим керівником щодо створення першого розділу	16.10.20 – 18.10.20р.	
4.	Розробка розділу 1	19.10.20 – 29.10.20р.	
5.	Розробка розділу 2	30.10.20 – 10.11.20р.	
6.	Розробка розділу 3	11.11.20 – 21.11.20р.	
7.	Розробка розділу 4	22.11.20 – 28.11.20р.	
8.	Висновки та оформлення пояснювальної записки дипломної роботи	28.11.20 – 02.12.20р.	
9.	Підписання необхідних документів у встановленому порядку	03.12.20 –10.12.20р.	
10.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломної роботи	11.12.20 – 21.12.20р.	

7. Дата видачі завдання _____

Керівник дипломної роботи _____ Холявкіна Т.В.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Деяк А.Ю.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Технологія сучасних односторінкових веб-додатків Single Page Application» викладена на 71 сторінках, містить 20 рисунків, 1 таблицю, 6 літературних джерел.

Об'єкт дослідження: односторінкові веб-додатки Single Page Application.

Мета роботи: створити сучасний односторінковий веб-додаток для ефективного використання в сфері бізнесу за допомогою JavaScript фреймворку Angular 10.

Предмет дослідження: створення та аналіз односторінкових веб-додатків за допомогою JavaScript фреймворка Angular 10.

Ключові слова: SINGLE PAGE APPLICATION, ОДНОСТОРІНКОВИЙ ВЕБ-ДОДАТОК, ФРЕЙМВОРК ANGULAR 10, СУЧАСНІ ТЕХНОЛОГІЇ СТВОРЕННЯ САЙТІВ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

SPA – Single Page Application

AJAX – Asynchronous Javascript and XML

API – Application Programming Interface

URL – Uniform Resource Locator

JS – JavaScript

CLI – Command Line Interface

UI – User Interface

UX – User Experience

JSON – JavaScript Object Notation

HTTP – HyperText Transfer Protocol

NPM – Node Package Manager

CLI – Angular Command Line Interface

AOT – Ahead Of Time

ВСТУП

В сучасному світі веб-технології розвиваються з неймовірною швидкістю. З кожним днем кількість нових сайтів збільшується. На сьогодні інтернет-ресурс є універсальним засобом для реклами, просування компанії, є наочним показником здатності використовувати інноваційні технології для підвищення якості роботи та є універсальним засобом для розроблення односторінкових веб-додатків.

Доступність веб-сайтів і платформонезалежність роблять їх дуже цікавими для розробників. Сьогодні, розвиток веб-технологій дозволяє будувати не лише веб-сайти у звичайному сенсі цього слова, коли веб-сайт складається зі сторінок, а також інтерактивні сайти – веб-сервіси, які можна назвати повноцінними додатками. Розробники веб-сайтів все більше і більше створюють онлайн версії популярних додатків, оскільки веб-сервіси становлять значну конкуренцію нативним додаткам.

Односторінковий веб-додаток SPA (Single Page Application) – це веб-сайт або веб-додаток, який використовує єдиний HTML документ, як оболонку для всіх веб-сторінок й організовує взаємодію з користувачем через динамічно завантажувані HTML, CSS, JavaScript, зазвичай за допомогою AJAX. Ідея SPA виникла ще коли, з’явились AJAX та JavaScript і з того часу веб-сайти почали розробляти більш динамічними та інтерактивними, будучи вже не тільки статичними сторінками. Мабуть кожному розробнику веб-сайтів рано чи пізно спадала думка щодо ідеї веб-додатка, який буде завантажуватись лише один раз, а вся взаємодія користувача буде відбуватись за рахунок асинхронних запитів та скриптів. Проте обсяг роботи, який потрібно виконати не вартий цього, адже для створення веб-сайту за концепцією SPA потрібно докласти значних зусиль. Щоб полегшити розробку за концепцією SPA може допомогти використання сучасних технологій, бібліотек, фреймворків та інструментів, що були для цього розроблені останнім часом. Завдяки цим технологіям, можна будувати ефективні односторінкові веб-додатки, витрачаючи на це час, майже еквівалентний, який необхідний на розробку традиційного нативного додатка.

Метою даної роботи є створення односторінкового веб-додатка з використанням JavaScript та фреймворка Angular 10, збір теоретичних відомостей про концепцію односторінкового веб-додатку та типові особливості його архітектури, огляд наявних популярних та відомих SPA додатків, технологій, що були використані при їх розробці, огляд інструментів та фреймворків, що використовуються при їх розробці.

Актуальність теми набирає обороти з кожним днем. Популярність розробки SPA набирають, використовуючи найчастіше Javascript. Тут нема чому дивуватися, через те, що SPA демонструє високу продуктивність та швидкість розробки, а також відмінно працює на всіх пристроях. Більшість розробників вважають популярними саме SPA програми. Аналізуючи веб технології 2020 року, можна зробити висновок, що в 2021 році, кількість SPA тільки буде збільшуватись, оскільки більшість користувачів все активніше використовують гаджети.

РОЗДІЛ 1

КОНЦЕПЦІЯ ОДНОСТОРІНКОВОГО ВЕБ-ДОДАТКУ

1.1 Огляд концепції

Створення SPA веб-додатків – це один із найбільш зростаючих трендів у сфері розробки програмних продуктів. Даний підхід до проектування веб-проектів дає можливість запускати складні програмні інструменти, що складаються всього з однієї HTML сторінки, на будь-яких пристроях, де є встановлений веб-браузер. Створення односторінкових веб-додатків дозволяє реалізувати цифрове бізнес-рішення у вигляді швидкого і стабільного веб-сервісу, інтерфейс якого нагадує не звичайний веб-сайт, а справжню прикладну програму.

Односторінкові SPA веб-додатки – це окремий випадок веб-додатків, які повсюдно приходять на зміну класичним веб-сайтам чи лендінгам. Основною відмінністю веб-додатків від традиційних сайтів полягає в тому, що користувач одержує не статичну інформацію у вигляді графіків або таблиць чи тексту, а динамічно згенеровані структуровані дані. Ці динамічно згенеровані дані можуть являти собою все ту саму графіку, текст чи таблиці, що отримано при інтерактивній взаємодії клієнта та сервера, де зберігається веб-додаток.

На відміну від традиційного сайту, веб-додаток – це не набір HTML документів, а цілий програмний комплекс, який завантажується на клієнтський пристрій після попередніх обчислень. Робиться це на підставі інформації та даних користувача, яка зберігається в базі даних веб-додатку. Ці обчислення зазвичай проводяться на стороні клієнта та сервера.

Суть SPA архітектури полягає в тому, що користувальницькі елементи, які потрібні для роботи програмного забезпечення знаходяться на одній веб-сторінці. Всі елементи користувальницького інтерфейсу завантажуються при ініціалізації, а у при взаємодії користувача з користувальницьким інтерфейсом, завантажуються

додаткові скрипти (модулі). Будь-яка призначена активність для користувача фіксується для зручності в навігації. Це дозволяє скопіювати посилання та відкрити веб-додаток на тому ж етапі взаємодії в іншому пристрої, браузері або вкладці.

SPA підхід дає можливість безпечно та швидко запускати найскладніші програмні комплекси на будь-яких гаджетах – від смартфонів і планшетів до персональних комп'ютерів, ноутбуків, а найголовніше не прив'язуючись до певної операційної системи. За допомогою цього підходу обчислення та значна частина апаратного навантаження спадає на сервер, в той час як на пристрої лише компоненти й модулі, які відповідають за побудову користувальницького інтерфейсу.

1.2 Традиційні й односторінкові веб-додатки

На сьогодні існують два способи створення веб-сайтів. Перший спосіб полягає у створенні традиційного веб-додатка, переважна частина логіки якого буде виконуватись на веб-сервері. Інший спосіб полягає у створенні односторінкового веб-додатка, логіка користувальницького інтерфейсу, якого виконується переважно в веб-браузері, а взаємодія з веб-сервером виконується за допомогою веб-API. Також існує і гібридний підхід, при якому в найпростішому випадку в рамках великого традиційного веб-додатку розміщуються одне або кілька повнофункціональних односторінкових веб-додатків, які побудовані за концепцією SPA підходу.

Традиційні веб-додатки необхідно використовувати в наступних випадках:

1. Веб-додаток повинен працювати в браузерах без підтримки JavaScript.
2. На стороні клієнта до веб-сайта застосовуються мінімальні вимоги, наприклад, використовуються тільки функції читання.
3. Команда розробників мало знайома з принципами розробки на JavaScript або TypeScript.

Односторінковий веб-додаток SPA необхідно використовувати в наступних випадках:

1. У веб-додатку необхідний повнофункціональний користувальницький інтерфейс.

2. У веб-додатку повинен надаватися веб-API для інших внутрішніх або загальнодоступних клієнтів.

3. Команда розробників повинна бути знайома з принципами розробки на JavaScript або TypeScript.

Для ефективної роботи з односторінковими веб-додатками потрібно набагато більший досвід в побудові архітектури та забезпеченні безпеки. Варто зазначити, що в односторінковому веб-додатку може ускладнитись налаштування процесів розгортання.

1.3 Коли слід вибирати односторінкові веб-додатки

У додатку необхідний повнофункціональний користувальницький інтерфейс.

На сьогодні односторінкові веб-додатки вміють підтримувати функції на стороні клієнта, які не вимагають перезавантаження сторінки під час навігації до різних розділів програми чи виконання будь-яких дій. Односторінкові веб-додатки підтримують часткові оновлення веб-сторінки, забезпечуючи збереження заповнених форм чи документів. Односторінкові веб-додатки швидше завантажуються, здійснюючи вибірку даних у фоновому режимі. Швидкість реагування на дії користувача зростає, оскільки повне перезавантаження сторінки виконується рідко. Односторінкові веб-додатки можуть бути налаштовані для роботи навіть при відсутності підключення до інтернет-мережі, що дозволить оновити графічний інтерфейс на стороні клієнта і згодом синхронізувати його з сервером при відновленні та підключенні. Якщо є необхідність реалізувати розширені функції крім стандартних можливостей HTML форм, то варто подумати над застосуванням односторінкових веб-додатків.

Односторінкові веб-додатки реалізовують повнофункціональну поведінку на стороні клієнта, в тому числі можливості перетягування і вставки, що набагато простіше в порівнянні з традиційними веб-додатками. В односторінкових веб-

додатках доволі часто реалізовуються можливості, які вже вбудовані в традиційні веб-додатки. Наприклад, коли є необхідність відображати в адресному рядку URL-адресу, що відповідає за певну дію. За допомогою цього користувач може додавати таку URL-адресу в закладки для того, щоб надалі мати можливість повернення до нього. Кнопки для навігації "Назад" та "Вперед" також повинні бути реалізовані, щоб мати можливість використовувати браузер з передбачуваними результатами.

У веб-додатку повинен надаватися веб-API для інших внутрішніх або загальнодоступних клієнтів.

Активне використання веб-API для запитів і оновлення даних під час роботи користувача з веб-додатком дозволяють односторінкові веб-додатки. У разі, коли підтримується використання веб-API іншими клієнтами, можливо кращим рішенням буде краще створити реалізацію односторінкового веб-додатку, який використає ці веб-API замість відтворення логіки на стороні сервера.

Команда розробників повинна бути знайома з принципами розробки на JavaScript або TypeScript.

Знання з принципами програмування та бібліотеками на стороні клієнта із застосуванням JavaScript чи TypeScript знадобляться для створення односторінкових веб-додатків. Команді розробників необхідно мати досвід написання сучасних додатків на мові програмування JavaScript за допомогою платформи односторінкових веб-додатків, таких як React, VueJs, Angular.

1.4 Коли слід вибирати традиційні веб-додатки

Веб-додаток повинен працювати в браузерах без підтримки JavaScript.

Веб-додатки, які необхідно щоб працювали в браузерах з повною відсутністю підтримки JavaScript або з обмеженою підтримкою, слід створювати із застосуванням робочих процесів традиційних веб-додатків. Або, як мінімум, слід реалізувати можливість перемикання на таку поведінку. Якщо нема можливості реалізації JavaScript на стороні клієнта, то вибирати модель односторінкового веб-додатку не рекомендується.

На стороні клієнта до веб-сайту застосовуються мінімальні вимоги, наприклад, використовуються тільки функції читання.

Існує певна частина користувачів багатьох веб-додатків, які можуть працювати тільки за допомогою функцій читання. Веб-додатки, які призначені тільки для читання, зазвичай набагато простіші ніж в яких реалізується управління станом. Наприклад, для реалізації пошукової системи цілком достатньо створити одну точку входу з текстовим полем та іншу сторінку для відтворення результатів пошуку. Якщо передбачено, що запити можуть виконувати анонімні користувачі, у зв'язку з чим на стороні клієнта практично не потрібно реалізовувати логіку. Схожим чином публічні додатки блогів чи системи управління контентом практично не мають функцій, що реалізуються на стороні клієнта та працюють переважно з контентом. Саме такі веб-додатки легко створювати в форматі традиційних серверних веб-додатків, які виконують логіку на веб-сервері й перетворюють HTML код для відображення в браузері. На сьогодні існує безліч фреймворків та бібліотек, за допомогою яких можна легко реалізувати власний блог чи сайт-візитку. Також варто зауважити, що кожна унікальна сторінка сайту має власну URL-адресу, яка індексується за допомогою пошукових систем. Така поведінка не вимагає додавання в веб-додаток окремої логіки, оскільки реалізується за замовчуванням.

Команда розробників не знайома з принципами розробки на JavaScript або TypeScript.

У разі, коли команда розробників мало знайома з програмуванням на JavaScript чи TypeScript, проте має досвід розробки серверних веб-додатків, то створення традиційних веб-додатків швидше за все займе набагато менше часу і буде набагато ефективніше. Якщо нема мати цілі навчитись створювати односторінкові веб-додатки або реалізувати надані ними можливості взаємодії з користувачем, більш продуктивним вибором для команд, знайомих з традиційними веб-додатками, стануть саме вони.

1.5 Коли слід вибирати нативний додаток

Односторінкові веб-додатки намагаються бути подібними до нативних додатків. При створенні програми, проект-менеджери чи розробники можуть задати собі питання: «Чому не створити нативний додаток, який зможе замінити односторінковий чи традиційний веб-додаток?». Річ у тому, що нативні додатки також мають свої недоліки, як і односторінкові чи традиційні веб-додатки. Односторінковий веб-додаток намагається перейняти все найкраще від нативного додатку та веб-сайту. Проте існують випадки, коли використання звичайного нативного додатку буде вигіднішим та більш необхідним.

Переваги нативних додатків:

1. Незалежність від інтернет з’єднання.
2. Продуктивність нативного коду.
3. Вільність у вибору мов програмування та інтерфейсів.
4. Використання нативних можливостей платформи.
5. Доступ до будь-якого обладнання пристрою.

Переваги односторінкових веб-додатків:

1. Можливість віддалених розрахунків.
2. Відсутність необхідності встановлення програмного забезпечення.
3. Не займає місце на жорсткому диску.
4. Багатоплатформність.

1.6 Таблиця прийняття рішень

У таблиці прийняття рішень представлені основні фактори, які слід враховувати при виборі між традиційним веб-сайтом, односторінковим веб-додатком та нативним додатком.

Табл.1.1.

Таблиця прийняття рішень

Фактор	Традиційний веб-сайт	Односторінковий веб-додаток	Нативний додаток
Знайомство команди з JavaScript або TypeScript	Мінімально	Обов'язкове	Не обов’язкове
Підтримка браузерів без скриптів	Підтримується	Не підтримується	Не потрібне
Мінімальні вимоги до додатка на стороні клієнта	Добре підходить	Надмірне	Мінімально
Складний повнофункціональний користувальницький інтерфейс	Обмежено	Добре підходить	Добре підходить
Встановлення	Не потрібно	Не потрібно	Потрібно
Інтернет з’єднання	Потрібно	Потрібно	Не обов’язкове
Навантаження на сторону клієнта	Середнє	Велике	Середнє
Доступ до обладнання клієнта та можливостей платформи	Не має	Не має	Є
Багатоплатформність	Є	Є	Не має
Гнучкість у вибору мови й інструментів розробки	Обмежена	Обмежена	Не обмежена

1.7 Дослідження існуючих веб-додатків

Google – це багатонаціональна компанія, яка найчастіше відома як пошукова система. Компанія була заснована в 1996 році Сергієм Бріном та Ларрі Пейджем, які на той час були студентами Стенфордського університету в Каліфорнії. Незважаючи на те, що компанія назвала себе такою ж назвою як і її пошукова система, переважна більшість доходу приходить саме від реклами через це, вона розгалужена на низку областей, таких як хмарні обчислення, програмне та апаратне забезпечення.

З початку заснування Google створив велику кількість різноманітної продукції, яка не зовсім пов’язана з головним продуктом компанії – пошуковою системою. Сьогодні Google виробляє сотні продуктів, якими користуються мільярди людей по всьому світу, починаючи від YouTube та Android, закінчуючи Gmail і, звичайно, Google Search. Переважна більшість продукції програмного забезпечення зроблені, як односторінкові веб-додатки.

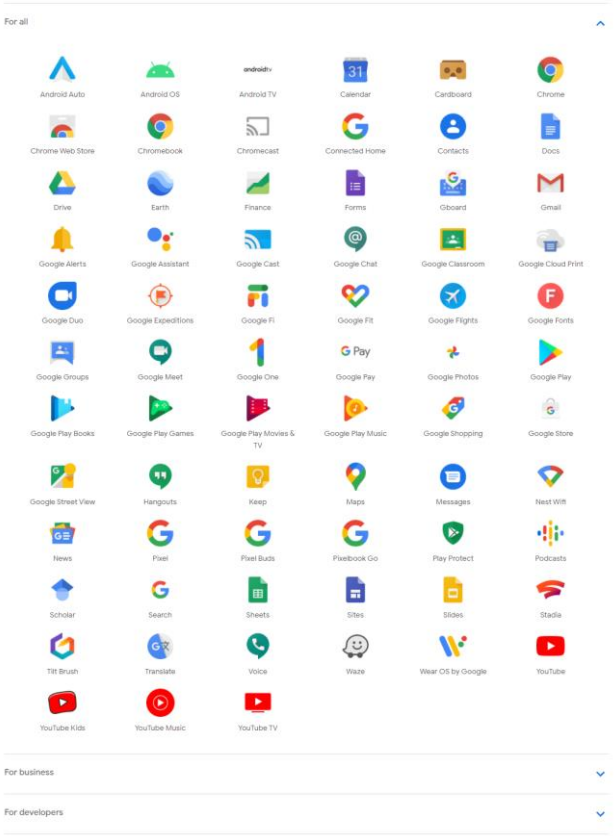

Рис. 1.1. Сервіси Google

1.7.1 Онлайн сервіси Google

Gmail – це безкоштовний сервіс електронної пошти, яку надає Google. Gmail багато в чому схожий на будь-яку іншу поштову службу. Користувач може отримувати та надсилати електронні листи, блокувати спам, створювати адресну книгу та виконувати інші завдання електронної пошти. Але він також має ще кілька унікальних функцій, які допомагають зробити його однією з найпопулярніших та найкращих електронних служб електронної пошти у світі. Gmail являється одним із найперших веб-додатків створених за концепцією SPA.

Цей сервіс був написаний на чистому HTML та JavaScript, використовуючи внутрішні бібліотеки Google. Створення Gmail було не легким процесом, оскільки майже не існувало ніяких створених фреймворків за концепцією SPA. Gmail виглядає доволі сучасно, оскільки Google постійно оновлює свої веб-сервіси.

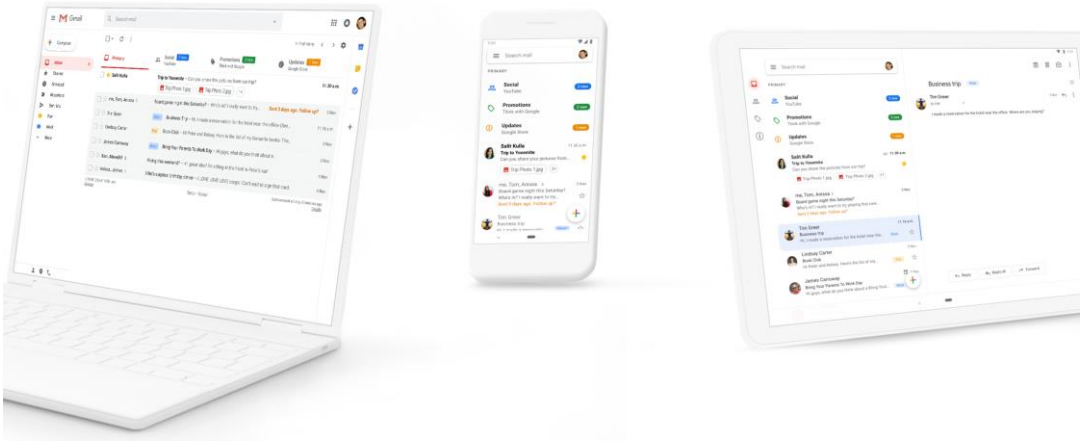

Рис. 1.2. Сервіс Gmail

Google Maps – один із найпопулярніших Google сервісів, яким користуються по всьому світі. Google Maps API є дуже зручним для використання, який часто використовується для інших компаній для роботи з картою. Цей односторінковий веб-додаток є безкоштовним як у використанні, як і більшість сервісів Google.

Google Maps доступний через веб-браузер або як додаток для мобільних пристроїв. Цей сервіс можна використовувати, щоб отримати покрокові вказівки, знайти інформацію про місцеві компанії та багато іншого. По суті цей сервіс являє собою карту та супутникові знімки планети Земля. Карти надають покрокові вказівки до місця призначення разом із 2D та 3D супутниковими знімками, а також інформацію про громадський транспорт. Карти також пропонують фотографічні знімки оточення, які показують справжні вулиці та околиці.

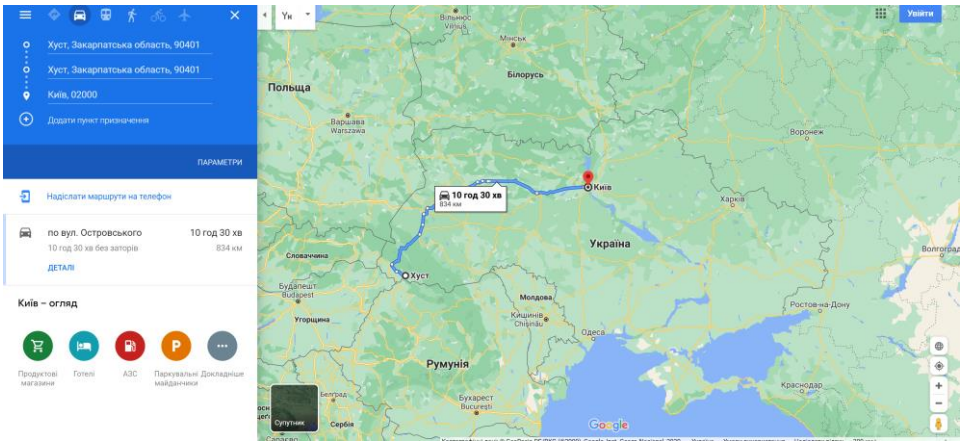

Рис. 1.3. Сервіс Google Maps

1.7.2 Онлайн сервіси Microsoft

Microsoft – це американська технологічна компанія. Вона була заснована Біллом Гейтсом і Полом Алленом в 1975 році, швидко виросла, перетворившись на найбільшу програмну компанію у світі. На сьогодні Microsoft все ще широко відома своїми програмними продуктами, але компанія також розробляє апаратне забезпечення та надає ряд хмарних послуг. Microsoft та Google, має безліч односторінкових додатків серед своїх онлайн сервісів. Найпопулярніші із них це онлайн версії офісного пакету такі як Microsoft Excel Online (рис. 1.4), сервіс хмарного сховища Microsoft OneDrive (рис. 1.5), Microsoft Word Online (рис. 1.6), Microsoft PowerPoint Online (рис. 1.7). Всі вище вказані додатки є тісно пов’язані між собою та дозволяють відкривати та редагувати офісні документи збережені в хмарному сховищі Microsoft OneDrive.

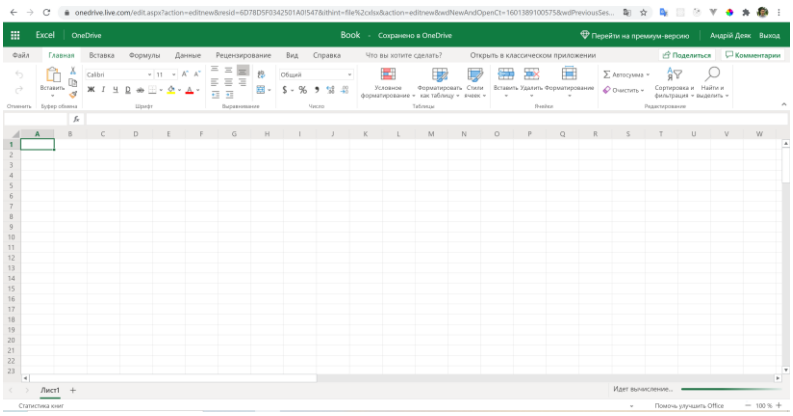

Рис. 1.4. Сервіс Microsoft Excel Online

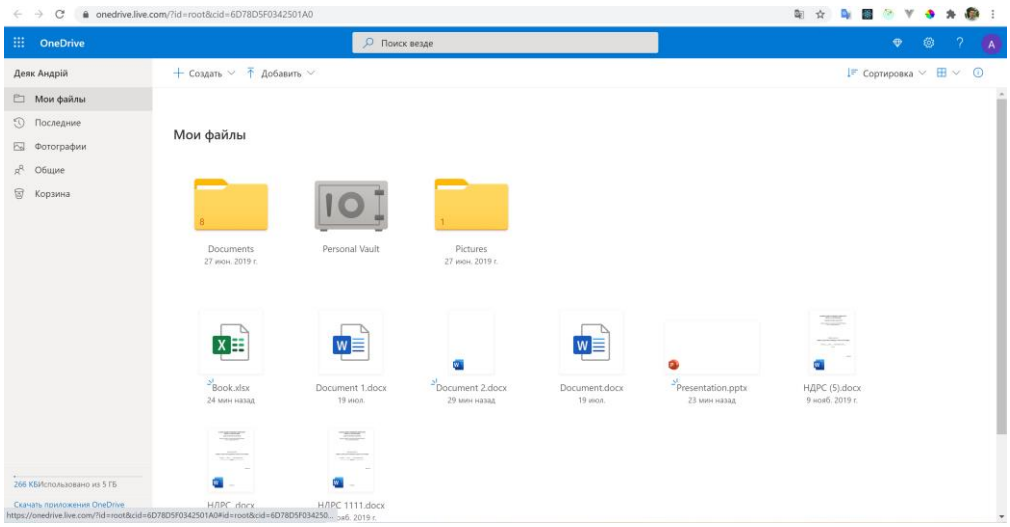

Рис. 1.5. Сервіс Microsoft OneDrive

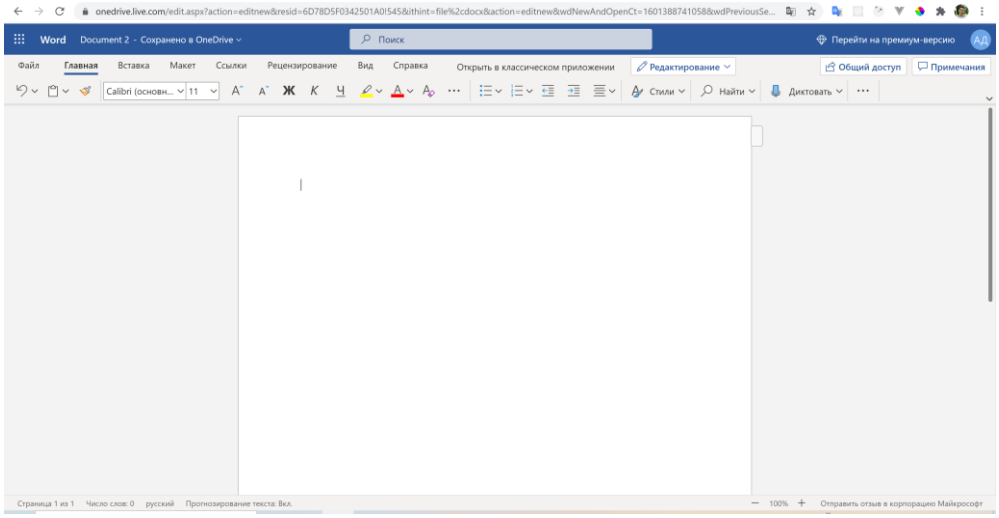

Рис. 1.6. Сервіс Microsoft Word Online

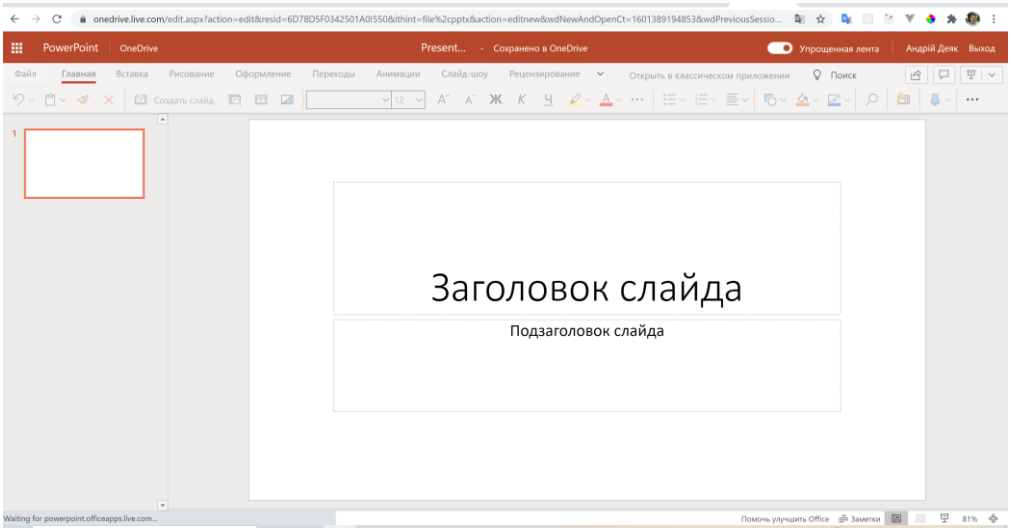

Рис. 1.7. Сервіс Microsoft PowePoint Online

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі був проведений аналіз основних положень концепції односторінкового веб-додатку. Проаналізовано в чому відмінність традиційних та односторінкових веб-додатків. Також розглянуті питання коли слід вибирати традиційні веб-сайти, односторінкові веб-додатки та нативні додатки. На основі аналізу щодо вибору між типами додаткві створено таблицю прийняття рішень. Проаналізовано існуючі веб-додатки та онлайн сервіси Google.

РОЗДІЛ 2

ДОСЛІДЖЕННЯ ТИПОВИХ ОСОБЛИВОСТЕЙ АРХІТЕКТУРИ ВЕБ-САЙТІВ

2.1 Розподілення ролей

На стороні сервера виконується більше роботи щодо відображення графічного інтерфейсу в традиційних веб-сайтах. У результаті доводиться створювати складні веб-сервери, які у свою чергу повинні справлятись з досить значним навантаженням. В односторінкових веб-додатках відображенням графічного інтерфейсу займаються саме клієнти, в результаті – розвантаження веб-сервера та на залишок тільки створення веб-API сервісної частини. Створення веб-сторінок та їх відтворення в односторінковому веб-додатку відбувається на стороні клієнта. Для розміщення односторінкового веб-додатку зі сторони серверу достатньо веб-API сервера сервісу та статичного веб-сервера. Для того, щоб краще зрозуміти як створюються односторінкові веб-додатки та традиційні веб-сайти було описано відповідальність клієнта та сервера.

Односторінковий веб-додаток

Клієнт:

1. Створення HTML, CSS, JS коду.
2. Створення логіки роботи графічного інтерфейсу.
3. Створення маршрутизації.

Сервер:

1. Налаштування статичного веб-серверу.
2. Налаштування веб-API сервісу.

Традиційний веб-сайт

Клієнт (браузер):

1. Певні незначні скрипти (для слайдерів, пагінації, меню, тощо).

2. Відтворення HTML, CSS, JS коду.

Сервер:

1. Налаштування веб-API сервісу.
2. Створення HTML, CSS, JS коду.
3. Створення логіки роботи графічного інтерфейсу.
4. Створення маршрутизації.

2.2 Інфраструктура

2.2.1 Компоненти

Типовий односторінковий веб-додаток зазвичай являє собою мінімальний набір інструментів, таких як: HTML, CSS, JS, TypeScript. Цей набір інструментів потребує чіткої структуризації, інакше ефективна розробка, масштабування і підтримка, стануть доволі не швидким та не легким процесом. Сторінки являються структурною одиницею в традиційних веб-сайтах. А в односторінкових додатках структуризація реалізується розділенням частини коду, а саме на компоненти.

Компонент – це практично механізм «plug-and-play» (підключи та грай) для об'єктів. За допомогою цього механізму можна конструювати складні сутності з різноманітною поведінкою. Компонент характеризується як певний об’єкт, який має ресурси й свою особливу поведінку, може містити в собі інші компоненти або залежати від них. Розділення на компоненти – широковідомий підхід, який використовується у майже у всіх фреймворках чи бібліотеках графічного інтерфейсу.

2.2.2 Модульність

Перше завантаження односторінкового веб-додатку може стати доволі довгим через те, що має велику кількість ресурсів, скриптів, так як вся логіка графічного інтерфейсу належить клієнту. Щоб позбутись довготривалого завантаження можна створити динамічне підвантаження ресурсів, тільки тоді

коли клієнту вони знадобляться. Саме ці ресурси потрібно розбивати на модулі, щоб динамічно підвантажувати та легше підтримувати код в майбутньому.

По мірі зростання веб-додатку, хорошою практикою вважається розділення веб-додатку на багато складових, файлів, так званих «модулів». Зазвичай модуль являє собою бібліотеку з функціями або клас.

Модуль – функціональний фрагмент програмного продукту, який оформлений у вигляді окремого файлу з кодом та призначений для використання і в інших програмах. Модулі вирішують одну з основних проблем при розробці програмного забезпечення – проблему масштабування програмних продуктів, що дозволяє розбивати складні задачі на менші, відповідно згідно принципу модульності. Також вирішує проблему багаторазового використання функціональності в інших частинах програмних продуктів. Зазвичай модулі і проектуються таким чином, щоб надати програмістам зручність масштабування та багаторазове використання функціональності.

Синтаксис модулів на рівні мови програмування JavaScript був відсутній протягом довгого часу. Оскільки, на початку епохи розробки веб-сайтів перші скрипти були не великими й простими, тому не було необхідності в модульності. Поступово скрипти ставали все більш і більш складними, тому спільнота вигадала кілька варіантів організації коду в модулі. Згодом з'явились бібліотеки для динамічної підвантаження модулів.

2.2.3 API модуль

Для виконання складних або пов’язаних з базою даних операцій на стороні сервера може стати хорошим рішенням створення окремого компонента. Переважна більшість таких дій потребують авторизації, яку можна реалізувати, як частину компонента. Ці компоненти можна винести в модуль і довантажувати в потрібний час.

2.2.4 Маршрутизація на стороні клієнта та сервера

Маршрутизація використовується майже на кожному веб-сайт. Маршрутизація у веб-розробці – це процес, який відповідає за визначення обробника для конкретної запитуваної сторінки. Найчастіше програмісти називають маршрутизацію як «роутинг». Маршрутизація обробляється по різному і може відрізнятись в реалізації в односторінкових веб-додатках та в традиційних веб-сайтах.

Якщо використовується клієнтська або серверна маршрутизація, то можна отримати доступ до неї з веб-сервера. На стороні веб-сервера зазвичай обробляються зміни URL-адрес в традиційних веб-сайтів. Впродовж розвитку JavaScript фреймворків були розроблені різноманітні методи управління маршрутизацією.

Маршрутизація – це спеціальний механізм управління URL-адресами, за рахунок якого можна підключити програмний код до певної URL-адреси. Цей механізм дозволяє зіставляти запити до додатку з певними ресурсами всередині програми. При зміні URL-адреси, наприклад, натиснувши на посилання, новий контент може бути наданий користувачеві.

Клієнтська сторона

Маршрутизація на клієнтській стороні реалізується за допомогою JavaScript. Кожен маршрут повинен бути налаштований на стороні клієнта, щоб здійнювати зміну стану програмного продукту. Зазвичай зміна стану програми призводить до зміни URL-адреси. Змінений стан програми – інша веб-сторінка, з новим контентом або запит на сервер для даних, які потрібно оновити.

Основна перевага клієнтської маршрутизації в порівнянні з серверною полягає у тому, що вся сторінка не оновлюється, а оновлюється тільки динамічні частини (елементи чи блоки) сторінки.

Мінуси:

- Сканування пошуковою системою є менше оптимізованими. Google, Yahoo, Bing чудово просуває в пошуку односторінкові веб-додатки, проте не так ефективно, як традиційні веб-додатки, які цілеспрямовані на сервер.
- У випадку односторінкового веб-додатку, за початковим запитом необхідно завантажити доволі велику частину сайту, через це саме перше завантаження може зайняти більше часу.
- Потрібно витратити більше часу з налаштуванням маршрутизації, можливо хорошим рішенням буде застосування певних бібліотек. Для створення маршрутизації на стороні клієнта необхідний допоміжний код, в той же час сторона сервера є стандартом, де не потрібно багато налаштувань.

Плюси:

- При зміні URL-адреси, плавні переходи й анімації легше реалізувати.
- В основному, маршрутизація між переглядами швидша, оскільки буде обробляєтись менше даних.

Серверна сторона

Під час роботи веб-сайту чи веб-додатку URL-адреса може змінюватись. При натисканні на посилання, яке буде вести на нову сторінку зі серверної сторони. Після цього користувачу буде поданий цілком нова, сформована веб-сторінка.

При запиті на стороні сервера буде відбуватись оновлення всієї сторінки. У цьому разі, відбудеться «GET» запит, який буде надісланий веб-серверу, після чого у відповідь веб-сервер відповість новою сформованою веб-сторінкою, повністю відкинувши стару веб-сторінку.

Мінуси:

- Зазвичай відображення сторінки займає більше часу ніж при клієнтській маршрутизації. Це може статись, у випадку, коли сторінка, яку потрібно відобразити, занадто велика, не є оптимізованою або у користувача можлива повільна швидкість обміну даними в інтернет-мережі.

- Оновлення цілої сторінки буде відбуватись при кожній новій запитуваній веб-сторінці. Це буде призводити до того, що буде оновлюватись непотрібний контент. Статичні блоки веб-сайту, такі як меню, підвал сайту та інші не потрібно оновлювати часто.

Плюси:

- Різноманітні реалізації серверної маршрутизації широко розповсюджені в інтернет-мережі.
- Пошукові системи оптимізовані для веб-сторінок, які створені на стороні сервера, так як серверна маршрутизація вже давно являється стандартом.

Звісно, що маршрутизація на стороні клієнта та на стороні сервера має і свої плюси та мінуси. Не існує ідеального рішення для управління маршрутизацією. Коли перед вибором стоїть клієнтська чи серверна маршрутизація, важливо прийняти оптимальне рішення виходячи з вимог.

2.2.5 Графічний інтерфейс

Вигляд веб-сайту надає розмітка сторінки HTML, каскадні таблиці стилів CSS та мова програмування JavaScript. Як відомо, в традиційних веб-сайтах CSS та HTML і JavaScript генерується на серверній стороні під час обробки запита, а в односторінкових веб-додатках всі скрипти відбуваються на клієнтській частині. Важливо, що для продуктивної роботи односторінкового веб-додатка необхідно уникати зайвих запитів до веб-сервера. Тому окрім основних HTML, CSS і JavaScript, які отримуються під час першого запуску веб-додатку, всі інші скрипти повинні створюватись на стороні клієнта. З періоду існування веб-сайтів до сьогодні для створення HTML, CSS, JavaScript на сервері було розроблено багато технологій і фреймворків, які зможуть полегшити процес розробки та підтримки. Проте, на сьогодні виникла потреба на такі технології на клієнтській стороні. Існує багато бібліотек і фреймворків, які полегшують процес створення графічного інтерфейсу за допомогою скриптів. Використання бібліотек і фреймворків – це необхідність для швидкого створення великого,

масштабованого, програмного інтерфейсу, який буде зручно та легко підтримувати.

2.2.6 Шаблони

Для вирішення задачі повторного використання та структуризації коду при створенні графічного інтерфейсу на будь-якому фреймворку хорошим рішенням вважається використання шаблонів.

При розробці веб-сайтів на стороні клієнта шаблони зазвичай являють собою набір HTML, CSS, JS файлів. Кожен подібний компонент може бути перевикористаний, змінюючи в ньому лише необхідні частини відповідно особливостям екземпляра. Можливості шаблонів напряму залежать від фреймворка, який буде обраний для написання програмного продукту. Шаблони підтримуються багатьма сучасними фреймворками, зокрема:

- Angular
- ReactJs
- VueJs

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі був проведений аналіз розподілу ролей щодо клієнта та сервера в традиційних веб-сайтах та односторінкових веб-додатках. Проаналізовано інфраструктуру побудови односторінкових веб-додатків. Також розглянуті питання компонентів, модульності, API модуля, маршрутизації на стороні клієнта та сервера, графічного інтерфейсу та використання шаблонів.

РОЗДІЛ 3

ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РОЗРОБКИ ВЕБ-ДОДАТКУ З ВИКОРИСТАННЯМ ANGULAR 10

3.1 Особливості Angular

Angular – це основа для побудови клієнтських додатків у HTML, CSS та Javascript, Typescript. Typescript – це мова програмування, яка компілюється в Javascript. Typescript дуже поширена мова програмування при використанні Angular – адже навіть сам Angular написаний на мові програмування Typescript.

Складні програми важко підтримувати тільки за допомогою Javascript та бібліотеки Jquery. Є необхідність в певному способі структуризувати програмний продукт. Ось чому на практиці часто використовується, такий фреймворк як Angular.

Angular – це фреймворк веб-додатку з відкритим кодом від Google (на базі TypeScript). Він розроблений спеціально для створення динамічних веб-додатків. За допомогою цього фреймворку можна створювати складні користувальницькі інтерфейси, не використовуючи інші плагіни, бібліотеки чи фреймворки. Angular використовується для розробки надсучасних веб-додатків на стороні клієнта, а саме односторінкових веб-додатків. Цей фреймворк має ряд інструментів і функцій, які спрощують розробку самих веб-додатків, одночасно гарантуючи чудові результати роботи. Оскільки, фреймворк Angular набув популярності серед розробників, що означає, що існує безліч інструментів, які можуть полегшити розробку програмних продуктів за допомогою цього фреймворку. Angular являється спадкоємцем іншого фреймворка AngularJS. Водночас Angular це не просто нова версія AngularJS, а принципово новий фреймворк.

При розробці односторінкових веб-додатків на фреймворку Angular

програміст не обмежений мовою TypeScript. При бажанні можна писатии програмні продукти на Angular за допомогою таких мов як Dart або JavaScript. Однак TypeScript все-таки є основною мовою для Angular. Остання версія Angular – Angular 10 вийшла в червні 2020 року.

3.2 Початок роботи з Angular

Щоб почати роботу з Angular потрібно встановити пакетний менеджер Node Package Manager та сервер Node.js в тому випадку, якщо вони відсутні на робочій машині.

Node.js – це в основному середовище виконання для виконання коду Javascript поза браузером. У цьому проекті ми в основному не використовуємо Node.js, але Node.js надає деякі інструменти, необхідні для побудови Angular проекту.

Для встановлення можна використовувати програмну установку Node.js. Разом з сервером вона також встановить і Node Package Manager. При цьому особливих знань для роботи з Node.js і NPM не вимагається. Після завершення встановлення потрібно переконатись, що встановлення було успішним. Щоб це зробити, можна відкрити консоль (командний рядок) та виконати в ньому наступну команду:

node –version

У випадку успішного встановлення в командному рядку буде показано встановлену версію Node.js. Мінімальною версією Node.js, необхідною для побудови Angular додатка, є версія 6.9.

Встановлення Angular CLI

Після встановлення Node.js ми можемо використовувати Node Package Manager або NPM для встановлення сторонніх бібліотек. Однією з бібліотек, яку необхідно встановити для проекту Angular, є Angular CLI або Angular Command Line Interface. Angular CLI – це інструмент інтерфейсу командного рядка, який використовується для ініціалізації, розробки, побудови та обслуговування програм Angular безпосередньо з командної оболонки. Це інструмент командного

рядка, який можна використовувати для створення нового проекту Angular або для створення нового шаблону для запуску нового проекту. Angular CLI також використовується для компіляції та для створення компонентів, модулів, класів та інших сутностей. Для того, щоб встановити Angular CLI потрібно відкрити консоль (командний рядок) та виконати в ньому наступну команду:

npm install -g @angular/cli

Вище команда встановить пакет Angular CLI у якості глобального модуля, через це надалі при створенні нових проектів Angular, Angular CLI не буде потрібно встановлювати знову. Після завершення встановлення потрібно переконатись, що встановлення було успішним. Це можна перевірити, відкривши командний рядок та ввести:

ng –version

У випадку успішного встановлення в командному рядку буде показано встановлену версію Angular CLI. Наступним кроком буде встановлення самого проекту за допомогою команди із Angular CLI:

ng new project

Ця команда генерує купу файлів і папок для налаштування та запуску Angular проекту, а потім використовує NPM для завантаження сторонніх бібліотек.

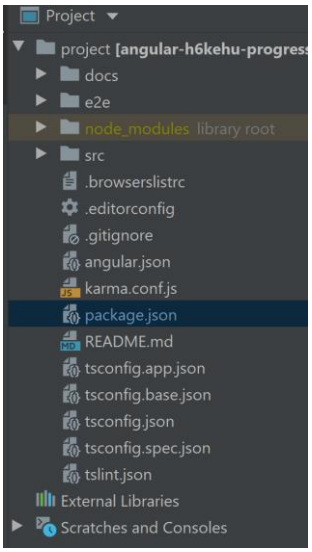

Рис. 3.1. Базова структура проекту

Angular CLI рекомендує створювати новий проект Angular за допомогою цієї команди, що дозволяє легко створити базовий програмний продукт, який вже працює, прямо зараз. Після завершення команди ми можемо побачити базову структуру нашого проекту (рис. 3.1).

3.3 Огляд базового компонента

Компоненти є найпростішими будівельними блоками фреймворку Angular. Angular містить дерево компонентів. Кожна програма Angular має як мінімум один базовий компонент. Ми можемо мати один або кілька компонентів у кожному Angular проекті, у великих додатках ми маємо десятки компонентів. Якщо зайти в папку src / app, то ми можемо побачити файли для базового компонента: «app.component.html», «app.component.scss», «app.component.ts», «app.component.spec.ts».

Компонент в контесті Angular – декоратор, який позначає клас, як Angular компонент та надає метадані конфігурації, що визначають, як компонент повинен оброблятись, створювати екземпляри та використовуватись під час виконання коду.

Відкривши файл «app.component.ts» можна побачити наступний код:

```
import { Component } from "@angular/core";
@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]})
export class AppComponent {
  title = "Andrew project";
}
```

Спочатку у файлі визначається директива import, яка імпортує функцію-декоратор @Component з модуля @angular/core. Під імпортом функції-декоратора

@Component прописується сама функція-декоратор @Component, яка зв’язує метадані з класом компонента AppComponent. В функції-декоратор @Component визначається параметр selector (відбірник), який буде відповідати за представлення компонента. За допомогою відбірника (selector) можна використовувати компоненти в різних місцях коду проекту. Сам відбірник являє собою ідентифікатор компонента в проекті. Під ним визначається параметр template (шаблон), який відповідає як саме потрібно компонент визначати з візуальної сторони. В кінці експортується клас самого компонента AppComponent, в якому саме визначається змінна title – в даному випадку це «Andrew project».

3.4 Створення модуля веб-додатку

@NgModule – декоратор, який доданий в Angular 2. Цей декоратор визначає сутність клас, як модуль Angular. Модулі Angular допомагають розширювати додаток, розбиваючи його на частини (модулі), які взаємодіють між собою і представляють в кінці цілісний веб-додаток. Модуль – це упаковка чи інкапсуляція частини функціоналу програмного продукту. Модулі потрібно проектувати з урахуванням багаторазового використання, щоб вони не залежали від конкретної реалізації програми. За допомогою модульної структури програмного продукту можна довантажити і задіяти саме ті модулі, які необхідні для продуктивної роботи програми. В Angular додатку завжди існує, як мінімум один головний модуль, який знаходиться в src / app / app.module.ts та має наступний вигляд :

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";
import { FormsModule } from "@angular/forms";
import { AppComponent } from "./app.component";
```

```
@NgModule({
imports: [ BrowserModule, FormsModule ],
declarations: [ AppComponent ],
bootstrap:[ AppComponent ]})
export class AppModule { }
```

Цей модуль називається AppModule, який буде початковим місцем для програмного продукту. Спочатку у файлі визначається директива import, яка імпортує декілька необхідних нам модулів. BrowserModule потрібен для роботи з браузером. Також імпортується фунцкія-декоратор NgModule, яка буде відповідати за створення даного модуля. FormsModule – допоміжний модуль, який ми надалі застосуємо для налаштування роботи форм. І в кінці, імпортується базовний компонент AppComponent.

3.5 Запуск веб-додатку

Щоб вказати Angular, як саме запустити односторінковий веб-додаток потрібно налаштувати файл main.ts:

```
import { AppModule } from "./app/app.module";
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

main.ts – це не модуль, а простий скрипт-файл, який виконується першим при запуску програмного продукту і може мати будь-яке інше ім'я файлу. Веб-додаток Angular може запускатись різними способами, але коли потрібно запустити програмний продукт в браузері, то є певний спосіб завантаження програми, який визначається в @angular/platform-browser-dynamic.

В папці src також визначено файл, який називається polyfills.ts. Цей файл містить поліфіли, які необхідні Angular і завантажується перед запуском додатка. Також можлива опція додавання власних поліфілів в даний файл.

Поліфіл – це програмний код, який додає в старі браузери підтримку можливостей, які в сучасних браузерах є вбудованими.

3.6 Створення головної сторінки

Розглянемо створення головної сторінки програмного продукту Angular. Якщо розглянути файл index.html, що знаходиться в папці src, то головна сторінка нашого односторінкового веб-додатку буде в собі містити наступний код:

```
!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <base href="/">
 <title>MyDreamApp</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

В елементі <body>…</body> визначено елемент <app-root> в який буде завантажуватись веб-додаток.

3.7 Визначення конфігурації

TypeScript є основною мовою для розробки додатків Angular. Це суперсет JavaScript з підтримкою під час проектування для забезпечення типізації. Браузери не можуть виконувати TypeScript безпосередньо. TypeScript повинен бути "переведений" у JavaScript за допомогою компілятора tsc, який вимагає певної конфігурації.

Angular містить декілька файлів конфігурації TypeScript. У кореневому файлі tsconfig.json вказуються базові параметри компілятора TypeScript та Angular, які успадковують усі проекти в робочій області.

tsconfig.json виглядає наступним чином:

```
{
 "files": [],
 "references": [
 { "path": "./tsconfig.app.json" },
 { "path": "./tsconfig.spec.json"}
]}
```

tsconfig.app.json виглядає наступним чином:

```
{
  "extends": "./tsconfig.base.json",
  "compilerOptions": {
  "outDir": "./out-tsc/app",
  "types": []
  },
  "files": [
   "src/main.ts",
   "src/polyfills.ts"
 ],
```

```
  "include": [
   "src/**/*.d.ts"
 ]}
```

tsconfig.base.json виглядає наступним чином:

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "importHelpers": true,
    "target": "es2015",
    "module": "es2020",
    "lib": ["es2018", "dom"]}}
```

Вище описані файли визначають налаштування для компілятора TypeScript. CompilerOptions – це параметр, який встановлює налаштування для компіляції веб-додатку. А параметр files в свою чергу, визначає, які файли потрібно скомпілювати. У наших налаштуваннях саме файли main.ts, який підтягує всі інші файли програми й файл поліфілів polyfills.ts буде скомпільований.

Angular.json

Angular CLI використовується для компіляції веб-додатка, тому нам потрібно описати налаштування компіляції за допомогою файлу angular.json.

angular.json виглядає наступним чином:

```
{
"version": 1,
"projects": {
  "project": {
    "projectType": "application",
    "root": "",
    "sourceRoot": "src",
    "architect": {
      "build": {
        "builder": "@angular-devkit/build-angular:browser",
        "options": {
          "outputPath": "dist/project",
          "index": "src/index.html",
          "main": "src/main.ts",
          "polyfills": "src/polyfills.ts",
          "tsConfig": "tsconfig.json",
          "aot": true
              }
            },
       "serve": {
         "builder": "@angular-devkit/build-angular:dev-server",
         "options": { "browserTarget": "project:build" }
              }
            }
           }},
          "defaultProject": "project"
        }
```

На самому початку файлу визначається параметр version. Version визначає, яка версія конфігурації проекту на даний час. Projects – параметр, який буде визначати налаштування для кожного проекту. Можна побачити, що у даному прикладі тільки один проект, який називається «project».

Наступні параметри проекту:

- projectType – може приймати одну з опцій «application» або «library». У випадку з «application» додаток може працювати самостійно у браузері, тоді як з опцією «library» це не можливо.
- root – вказує шлях на папку файлів проекту щодо робочого середовища. Якщо буде поставлене пусте значення, то це буде відповідати кореневій папці проекту. У даному прикладі каталог проекту та робоче середовище збігаються.
- sourceRoot – визначає кореневу папку файлів з кодом. У даному прикладі – це папка src, де розміщені всі файли програмного продукту.
- architect визначає налаштування для побудови проекту. Architect – це інструмент, який Angular CLI використовує для виконання складних завдань, таких як тестовий запуск і компіляція проекту. Architect – це оболонка, яка запускає вказаний builder для виконання заданого завдання, відповідно до цільової конфігурації. У файлі package.json є визначені команди serve та build і відповідно для кожної з них в секції architect існують свої налаштування.

Для кожної команди architect повинен задаватись параметр builder, який є інструментом для побудови проекту. Для команди build в опції builder встановлено значення «@ angular-devkit / build-angular: browser» – дана опція буде вказувати на те, щоб побудувати проект під опції браузера. А для команди «serve» встановлено значення «@ angular-devkit / build-angular: dev-server» – дана опція буде вказувати на те, щоб запустити веб-сервер і розгорнути на ньому скомпільований додаток.

Для побудови файлів існує параметр «options». Для команди "build" тут визначені наступні опції:

- outputPath – шлях, по якому буде публікуватись скомпільований програмний продукт.
- index – шлях до головній сторінки програмного продукту.
- main – шлях до основного файлу програми, з якого починається запускатись додаток Angular.
- Polyfills – шлях до файлу поліфілів.
- tsConfig – шлях до файлу конфігурації TypeScript.
- aot – вказує, чи буде використовуватися компіляція AOT (Ahead Of Time). В даній опції налаштовано значення true, що свідчить про те, що компіляція AOT використовується.

Angular AOT (Ahead Of Time) компіляція – це нова функція, яка додана до Angular для значного підвищення продуктивності. Це спосіб підвищити продуктивність, виконуючи більшу частину роботи під час побудови проекту.

Для команди "serve" визначена опція «browserTarget», яка містить посилання на конфігурацію для команди «build – "project: build"». Це означає, що дана команда використовує такі самі налаштування, як і команда build. Опція defaultProject визначає проект за замовчуванням. В нашому випадку це наш єдиний проект.

У випадку використання TypeScript у зв’язку з Angular і Angular CLI для компіляції, то файли tsconfig.json, tsconfig.app.json, package.json, tsconfig.base.json, angular.json по суті будуть присутними в кожному проекті. Ці файли можна переносити з проекту на проект з мінімальними змінами. Для прикладу, у файлі angular.json замість назви проекту "project" буде відповідна назва нового проекту. У файлі package.json можна буде задати інші версії пакетів, якщо попередні версії застаріли чи змінились. Можна змінити назву проекту чи версію налаштування, а також Angular CLI чи TypeScript, проте в цілому загальна організація буде такою самою.

3.8 Запуск проекту

Після вище описаних налаштувань проекту, можна запускати проект в браузері. Для цього в командному рядку перейдемо до папки проекту і виконаємо команду «ng serve»:

```
D:\training\Angular\diploma\project>ng serve

chunk {main} main.js, main.js.map (main) 59.6 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 12.8 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.62 MB [initial] [rendered]
Date: 2020-09-30T20:48:49.394Z - Hash: c8b7a72408716f9a02c5 - Time: 7977ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.
```

Рис. 3.2. Запуск проекту

Консольний вивід виведе на екран файли якого розміру створені. Також можна побачити адресу, за якою запущений тестовий веб-сервер – по замовчуванню це «http://localhost:4200». Якщо передати додатковий параметр «--open», то Angular CLI автоматично відкриє браузер із запущеним додатком. І вже після цього можна взаємодіяти з додактом, який працює в браузері:

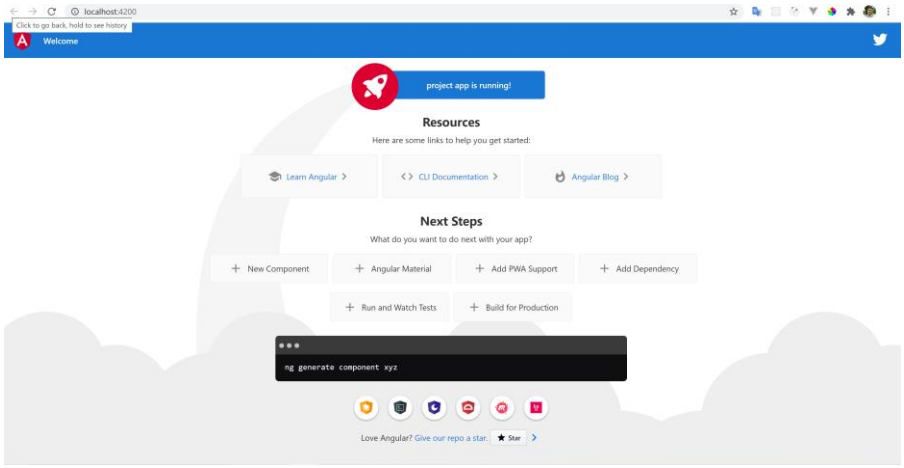

Рис. 3.3. Запущений проект в браузері

У випадку зміни коду в додатку, Angular CLI майже миттєво перекомпілюються і оновить веб-додаток в браузері.

ВИСНОВОК ДО РОЗДІЛУ 3

В розділі було досліджено особливості розробки веб-додатку з використанням Angular 10. Вияснені, які харатерні особливості для Angular фреймворку. Розглянуто питання про те, як розпочати роботу з Angular. Оглянуто базовий компонент Angular. Також розгянуті питання щодо ствrення модуля веб-додатку, головної сторінки, визначення конфігурації та запуску проекту.

РОЗДІЛ 4

СТВОРЕННЯ ОДНОСТОРІНКОВОГО ВЕБ-ДОДАТКУ З ВИКОРИСТАННЯМ ANGULAR 10

4.1 Описання односторінкового веб-додатку

Цей розділ присв'ячений реалізації працездатного односторінкового веб-додатку для мобільних та десктопних пристроїв. В результаті буде створений сучасний односторінковий веб-додаток для ефективного використання в сфері бізнесу, написаний за допомогою JavaScript на фрейморку Angular 10. Даний веб-додаток зможе допомогти залучити більше нових клієнтів, які займаються бізнесом, а саме бізнесом роздрібної торгівлі. Створений односторінковий веб-додаток може бути використаний, як рекламний продукт такого характеру, що користувачам невідмінно захочеться поділитися інформацією про нього з партнером по бізнесу в сфері продаж.

4.2 Фукціональні вимоги односторінкового веб-додатку

Даний веб-додаток буде розробелний для компаній, які займаються бізнес-аналізом світових брендів. Цей додаток буде являти собою інтерактивну панель для управління модної роздрібної торгівлі сучасних світових брендів, таких як «H&M», «ZARA», «UNIQLO». Ця інтерактивна панель буде зображена у вигляді динамічних графіків різного типу та панеллю управління, за допомогою якої користувач зможе фільтрувати дані для своїх потреб. Панель управління буде скаладатись з наступних фільтрів:

- Роздрібні торговці («Retailers»).
- Категорія («Category»).
- Період часу («Time frame»).

В фільтрі роздрібних торговців будуть доступні наступні опції:

- «H&M».
- «ZARA».
- «UNIQLO».

Інші роздрібні торговці, такі як «Mango», «Nike», «Asos», «Gucci», «Victoria Secret», «Ralph Lauren» будуть доступні тільки після підписки на преміум аккаунт системи. На преміум аккаунті будуть доступні дані про більше 3000 розрібних торговців.

В фільтрі категорії будуть доступні насупні опціїї:

- Всі категорії (без підкатегорій).
- Сукні.
- Футболки.

Інші категорії, такі як аксесуари, сорочки, блузки, брюки, джинси, трикотажні вироби, верхній одяг, взуття, шорти, спідниці будуть доступні тільки після підписки на преміум аккаунт системи. На преміум аккаунті буде доступно більше 30 інших категорій.

В фільтрі період часу будуть доступні наступні опції:

- За останній тиждень («Last week»).
- За останній місяць («Last month»).

Також буде існувати опція як «нестандартний період» («Custom period»), який буде доступний тільки після підписки на преміум аккаунт системи. В залежності від вибраного фільтру буде відображатись інформація для користувача.

Також в панелі управління буде знаходитись статистичні дані, які будуть вираховуватись на основі вибраних фільтрів. Статистичні дані будуть представлені у вигляді трьох елементів користувальницького інтерфейсу, які будуть вираховувати:

- Кількість нових товарів («New ins»).
- Найвищу знижку («Highest avg discount»).
- Найвищу найчастішу ціну («Highest most frequent price»).

Під фільтрами та під статистичними даними на панелі управління буде відображатись декілька графіків і таблиця:

- Таблиця структури цін («Price structure»).
- Графік асортиментів («Assortment Mix»).
- Графік нових товарів («New ins»).
- Середня ціна по категоріях («Average price – Average price per category»).
- Еволюція знижки для кожного бренду («Discount evolution – Discount structure per brand») – для преміум аккаунта.
- Структура кольорів для кожного бренду («Colors – Colors structure per brand») – для преміум аккаунта.
- Розпродаж для кожного бренду – («Sell out – Sell out structure per brand») – для преміум аккаунта.
- Склад запасів для кожного бренду – («Stock – Stock structure per brand») – для преміум аккаунта.
- Структура тканин для кожного бренду – («Fabrics – Fabrics structure per brand») – для преміум аккаунта.

Дана версія односторінкового веб-додатку розрахована на звичайного користувача системи. У цьому випадку інформація, яка назначена для преміум аккаунта буде відображатись як недоступна.

4.3 Підготовка моделі для клієнта

Моделювання даних – це процес дослідження та розроблення структур, які орієнтовані на дані. Подібно до інших моделюючих артефактів моделі даних можуть бути використані для різних цілей, починаючи від концептуальних моделей високого рівня і закінчуючи фізичними моделями даних. З точки зору об'єктно-орієнтованої розробки моделювання даних концептуально схоже на моделювання класів. За допомогою моделювання даних можна ідентифікувати типи сутностей, а за допомогою моделювання класів – класи. Атрибути даних присвоюються типам сутностей аналогічно, як і призначаються операції та атрибути класам. Між сутностями існують асоціації, подібні до асоціацій між

класами – відношення, успадкування, агрегація – це всі часто застосовні поняття в моделюванні даних.

Традиційне моделювання даних відрізняється від моделювання класів, оскільки воно фокусується виключно на моделях даних, що дозволяє досліджувати поведінку моделі в цілому, а за допомогою моделі класів можна досліджувати тільки проблеми з даними. Через це розробники при моделюванні даних мають тенденцію набагато краще моделювати дані , ніж ті хто моделює класи.

Моделі даних можна ефективно застосовувати, як в проектах, так і на рівні підприємств. Архітекторам програмного забезпечення підприємств доволі часто доводиться створювати одну або декілька логічних моделей даних високого рівня, які відповідатимуть за відображення структур даних, що підтримують підприємство, моделі, які зазвичай називають інформаційними моделями підприємства або моделями даних підприємств. Інформаційна модель підприємства або модель даних для підприємства – це один із декількох поглядів, який архітектори програмного забезпечення підприємства можуть вибрати для покращення та підтримки. Моделі даних підприємств містить інформацію, яку команда проекту може ефективно використовувати, як набір обмежень, так і важливу інформацію про структуру своєї внутрішньої системи.

Модель даних відноситься до логічних взаємозв’язків та потоків даних між різними елементами даних, що беруть участь в інформаційній системі підприємства чи проекту. Вона також забезпечує спосіб отримання та зберігання даних. Моделі даних сприяють технічному розвитку і комерційному бізнесу, точно представляючи вимоги інформаційної системи та розробляючи відповіді, які необхідні для цих вимог. Моделі даних допомагають чітко уявити який формат та які дані потрібно застосувати для різних бізнес-процесів.

Модель даних – це абстрактна модель, яка встановлює відношення та зв’язки між елементами даних і стандартизує їх зв'язок один з одним. Термін модель даних може стосуватися для двох різних, проте тісно пов'язаних між собою концепцій. З однієї сторони це часто стосується абстрактної формалізації

відносин і об’єктів, які відтворені у певній області застосування. З іншої сторони, це стосується набору понять, що застосовуються при визначенні таких формалізацій, як атрибути, сутності, відносини, таблиці.

У програмній інженерії моделювання даних – це процес створення моделі даних за допомогою застосування формальних описів моделі даних із використанням методів для моделювання даних. Іншими словами, моделювання даних – це методика для визначення бізнес-вимог до бази даних. Архітектори програмного забезпечення часто цей процес називають моделюванням бази даних, оскільки модель даних буде реалізовуватись в базі даних.

Властивості моделі даних

Деякі важливі властивості даних, для яких необхідно виконати вимоги:

1. Властивості, пов'язані з визначенням:
   1.1. Ясність – характеризує єдино та чітко визначені дані.
   1.2. Актуальність – визначає корисність даних в контексті підприємства чи проекту.
   1.3. Узгодженість – визначає сумісне існування даних з різних джерел.
2. Властивості, пов'язані зі змістом:
   2.1. Точність – описує наскільки точні дані до істини.
   2.2. Своєчасність – характеризує ступінь актуальності даних та доступність, наявність в потрібний час.
3. Властивості, пов’язані як зі змістом, так і з визначенням:
   3.1. Доступність – визначає де, як і кому дані недоступні чи доступні (наприклад, обмежений доступ, безпека).
   3.2. Повнота – визначає скільки необхідних даних існує.
   3.3. Вартість – описує витрати, понесені при отриманні даних та надання їх для доступності.

У нашому випадку, модель даних може вказувати, що елемент даних, який представляє панель управління, складається з ряду елементів, що, в свою чергу, представляють бренди, колір для брендів, категорії, період часу.

Підготовка моделі для клієнту є важливою частиною проектування, оскільки правильно сформована модель буде легше обслуговуватись як на сервері, так і на клієнті.

Виходячи з фукціональних вимоги односторінкового веб-додатку, можна спроектувати модель даних. Модель даних для панелі управління буде являти собою об’єкт, в якому буде знаходитись наступні сутності:

```
export interface DashboardData {
  brands: SelectItem[];
  brandColors: { [key: string]: string };
  categories: SelectItem[];
  lastWeek: AvailableCategories;
  lastMonth: AvailableCategories;
}
export interface AvailableCategories {
  [key: string]: AvailableCategoriesData;
}
export interface AvailableCategoriesData {
  assortmentMix: AssortmentMix;
  priceStructure: PriceStructure;
  newIns: NewIns;
  statistic: {
    avgDiscount: Statistic;
    avgPrice: Statistic;
    newIns: Statistic;
  };
}
export interface Statistic {
  [key: string]: number;
}
```

```
export interface SelectItem {
  id?: string;
  value: string;
  locked?: boolean;
}
export interface PriceStructure {
  data: PriceStructureData[];
  displayedColumns: string[];
  measurement: string;
  currency: string;
}
interface PriceStructureData {
  [key: string]: string | number;
}
interface NewIns {
  data: { [key: string]: number[] };
  timestamps: string[];
}
export interface AssortmentMix {
  categoriesLabels: string[];
  data: { [key: string]: number[] };
  colors: string[];
}
export interface HiddenCharts {
  img: string;
  title: string;
  subTitle: string;
}
```

Варто зазначити, що моделі для полів «lastWeek», «lastMonth» будуть еквівалентними. А також «brands» і «categories» – еквівалентні моделі.

4.4 Створення веб-дизайну

Продуманий дизайн сайту є важливим чинником для успіху діяльності будь-якого бізнесу. Коли користувач вперше заходить на веб-сайт, то звертає увагу на веб-дизайн. Зручний, кваліфікований та грамотний дизайн веб-сайту безпосередньо виконує роль візитної картки і повинен вигідно представляти підприємство (компанію, бізнес) серед інших конкурентів. Якісний веб-сайт повинен повно та професійно відображати діяльність бізнесу компанії та забезпечувати зручність користування ресурсом і пропонувати послуги чи товари.

Прагнення до простоти – тенденція, яка спостерігається в сфері цифрових технологій в продовж довгого часу. Оскільки, постійно зростає попит на веб-сайти, односторінкові веб-додатки і лендінги, зростає і прагнення спростити веб-сайт максимально. Інструменти для візуального оформлення, розробки веб-сайтів стають все більш простішими. Секрет лаконічного UI/UX дизайну полягає не тільки в простому віртуальному каркасі з текстом і кнопками, а в корисному ресурсі з усвідомленням психології користувачів та їх реакції на продукт.

UI/UX (англ. User Interface / User Experience) – це створення механізму взаємодії користувача з веб-сайтом (веб-додатком, лендінгом, традиційним веб-сайтом) й візуальне втілення цього механізму. Перш ніж почати працювати над графічним інтерфейсом та розставленням елементів інтерфейсу, важливо мати доволі глибоке розуміння того, як підвести клієнта до цільової дії.

Професіонали напрямку UI/UX створюють інтерфейси, які в свою чергу суттєво полегшують взаємодію людей з будь-яким програмним продуктом. Вони знають, як утримати цільову аудиторію в програмного продукту, продумовують кожен майбутній крок користувача і роблять його більш зрозумілішим, сприяють прихильності до продукту. Інтерес користувача може швидко зникнути, якщо:

- перебирати навмання всі функції навігації в програмному продукті для виконання простої дії.
- витрачати багато часу на марні кліки в програмному продукті для отримання цільових даних;

- користуватись незручною панеллю управління, панеллю навігації (такі панелі повинні бути виконані бездоганно, так як вони найчастіше використовуються користувачем);

Простота, лаконічність, легкість та миттєве рішення цільового завдання – саме це потрібно сучасному користувачу програмного продукту. Головна мета комплексного дизайнерського рішення UI/UX – підвести користувача до конкретної логічної зупики, використовуючи найкоротшу дорогу. Дії користувача програмного продукту повинні виконуватися в максимально інтуїтивному інтерфейсі. Прихований функціонал за естетичним оформленням та саме оформлення повинні спростити користувачеві досягнення його цілей.

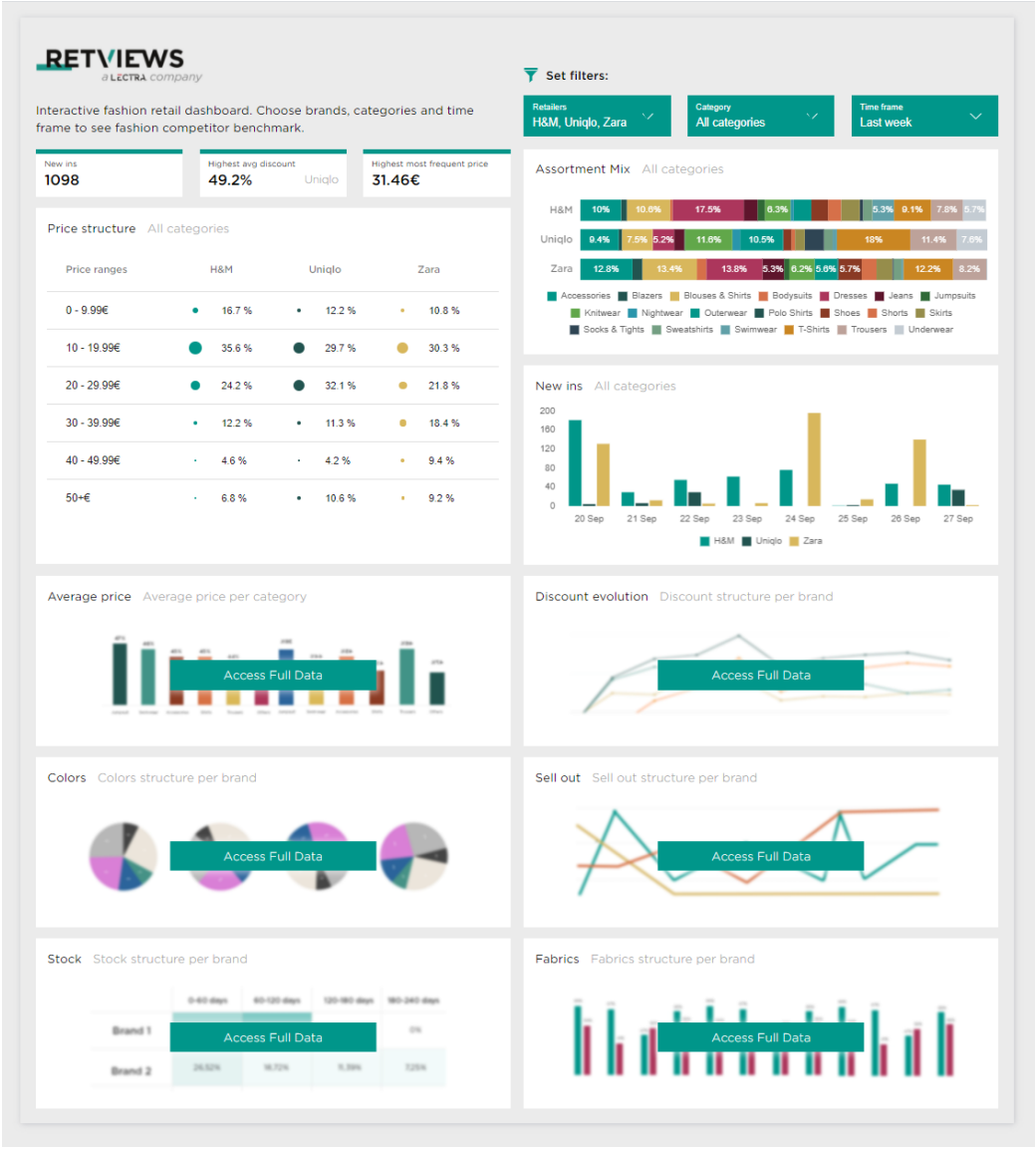

Рис. 4.1. Веб-дизайн панелі управління

Для успішної реалізації нашого майбутнього інтерфейсу для односторінкового веб-додатку я дотримувався наступних правил:

- Дотримувався архітектури, при якій логічно пов'язані елементи групуються разом (фільтри, графічні елементи, статистичні блоки).
- Вирівнював складові елементи інтерфейсу.
- Витримав один загальний стиль для фільтрів, графіків, кнопок.
- Всі елементи програмного продукту створені так, що взаємопов'язані та структуровані.
- Застосував акцентування уваги на певних кнопках чи блоках (не перевантажував зовнішній вигляд не потрібними елементами інтерфейсу, відволікаючими ілюстраціями);

Враховуючи найкращі практики по створенню зручного користувальницького інтерфейсу для користувача веб-додатку розроблений веб-дизайн (рис. 4.1). Лаконічно підібраний набір кольорів та естетичні тіні, відступи та розміри елементів надають дизайну привабливого вигляду.

4.5 Вибір бібліотек для реалізації проекту

Для програмної реалізації односторінкового веб-додатку панелі управління були застосовані наступні бібліотеки:

- Angular Material.
- Chart.js.

Material Design – це мова дизайну для десктопних та мобільних додатків, яка була розроблений компанією Google в 2014 році. Material Design спрощує розробникам налаштування UI, зберігаючи при цьому зручний інтерфейс додатків. Material Design дає можливість отримати гнучкість, добре організований формат, щоб висловити свій стиль та бренд.

Angular Material – це бібліотека компонентів, яка побудовані на принципі Material Design. Angular Material складається з наборів компонентів фреймворку Angular. На відміну від бібліотеки Bootstrap, яка надає компоненти, які можна використовувати будь-яким способом, Angular Material прагне забезпечити

послідовний та розширений інтерфейс. Водночас, в Angular Material існує можливість контролювати, як поводяться різні компоненти в залежності від установлених в них властивостей.

Chart.js – це бібліотека, яка призначена для створення графіків і діаграм. Вона дозволяє створювати діаграми і графіки будь-якого типу, а також відображати дані на певному діапазоні часу. Також в неї вбудовані засоби роботи з анімацією, що дозволяють ефектно видозмінювати графіки в залежності від нових даних, а також експериментувати з кольором.

Найчастіше звичайні користувачі погано сприймають дані, які представлені у вигляді таблиці чи тексту. Передусім, тому що це не так привабливо, але в набагато більшому ступені, бо в такому вигляді складніше зрозуміти цінність представлених даних. Навіть у випадку, якщо таблиця містить не багато даних, то все одно є шанс, що людина швидко перегляне її та упустить потрібну їй інформацію. Як найчастіше буває, коли користувач уважно перегляне тільки декілька елементів таблиці, які його найбільше зацікавлять. У випадку, коли ці ж дані були представлені у вигляді діаграми, то користувач без особливих зусиль зміг би побачити картину в цілому. Коли справа стосується діаграм, то набагато легше визначити якийсь факт чи тренд саме по динаміці графіка. Безсумнівно, через вище вказані причини – було прийняте рішення подавати більше даних для користувача у вигляді графіків.

4.6 Архітектура односторінкового веб-додатку

Angular – це платформа та фреймворк для побудови односторінкових клієнтських веб-додатків за допомогою HTML, CSS, JavaScript, TypeScript. Фреймворк реалізує основну та додаткову функціональність, як набір бібліотек TypeScript, які можна імпортувати у свої програмні продукти.

Архітектура програми Angular спирається на певні фундаментальні концепції. Основними будівельними блоками є модулі (NgModules), які забезпечують контекст компіляції для компонентів. NgModules групує

відповідний код у функціональні набори. Односторінковий веб-додаток Angular визначається набором NgModules.

Компоненти характеризують подання, які являють собою набори елементів користувальницького інтерфейсу, які Angular фреймворк може обирати та модифікувати відповідно до даних і логіки програми. Компоненти можуть використовувати сервіси, які можуть надати деякі функціональні можливості, не пов'язані безпосередньо з поданням даних. Сервіси можуть бути реалізовані в компонентах як залежності, роблячи код ефективним, модульним і багаторазовим.

Модулі, компоненти та сервіси – це по суті класи, що використовують декоратори. Декоратори в Angular надають метадані, за допомогою яких фреймворк може зрозуміти як саме опрацьовувати код. Метадані для класу сервісів надають інформацію, необхідну Angular, щоб зробити її доступною для компонентів за допомогою введення залежностей (Dependency injection).

Для односторінкового веб-додатку панелі управління достатньо одного модуля, оскільки в нас буде відображатись одна веб-сторінка. Розглянемо створений модуль, який називається AppModule з наступним виглядом:

```
import {HttpClientModule} from "@angular/common/http";
import {NgModule} from "@angular/core";
import {FormsModule, ReactiveFormsModule} from "@angular/forms";
import {MatButtonModule} from "@angular/material/button";
import {MatCheckboxModule} from "@angular/material/checkbox";
import {MatOptionModule} from "@angular/material/core";
import {MatDialogModule} from "@angular/material/dialog";
import {MatFormFieldModule} from "@angular/material/form-field";
import {MatSelectModule} from "@angular/material/select";
import {MatTableModule} from "@angular/material/table";
import {BrowserModule} from "@angular/platform-browser";
import {BrowserAnimationsModule} from "@angular/platform-browser/animations";
import {ChartsModule} from "ng2-charts";
```

```
import {AppComponent} from "./app.component";
import {LoaderComponent} from "./components/loader/loader.component";
import {MarketingDashboardPopoverComponent} from "./components/marketing-dashboard-popover/marketing-dashboard-popover.component";
import {MarketingDashboardComponent} from "./components/marketing-dashboard/marketing-dashboard.component";
import {NoResultsFoundComponent} from "./components/no-results-found/no-results-found.component";
@NgModule({
  declarations: [
    AppComponent,
    MarketingDashboardComponent,
    MarketingDashboardPopoverComponent,
    LoaderComponent,
    NoResultsFoundComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
    ChartsModule,
    MatFormFieldModule,
    MatOptionModule,
    MatSelectModule,
    BrowserAnimationsModule,
    MatCheckboxModule,
    MatTableModule,
    MatDialogModule,
    MatButtonModule,
    HttpClientModule
```

```
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {
}
```

Тег declarations реєструє наявні компоненти в AppModule модулі. Кожен компонент відповідає за свою логіку веб-додатку.

AppComponent – корневий компонент веб-додатку, в якому буде розміщений компонент MarketingDashboardComponent, що створений для відображення панелі управління.

MarketingDashboardPopoverComponent – компонент, відповідальний за спливаюче модальне вікно. Модальне вікно буде з'являтись у тому випадку, коли користувач натисне на один із елементів таблиці або графіку. У самому вікні відображено три товари, які будуть недоступні для перегляду для звичайного користувача. Щоб отримати доступ до даних про товар необхідно натиснути в модальному вікні «Access Full Data», після чого відбудеться переадресація на нову веб-сторінку. Щоб закрити модальне вікно потрібно натиснути за його межами. Модальне вікно виглядає наступним чином:

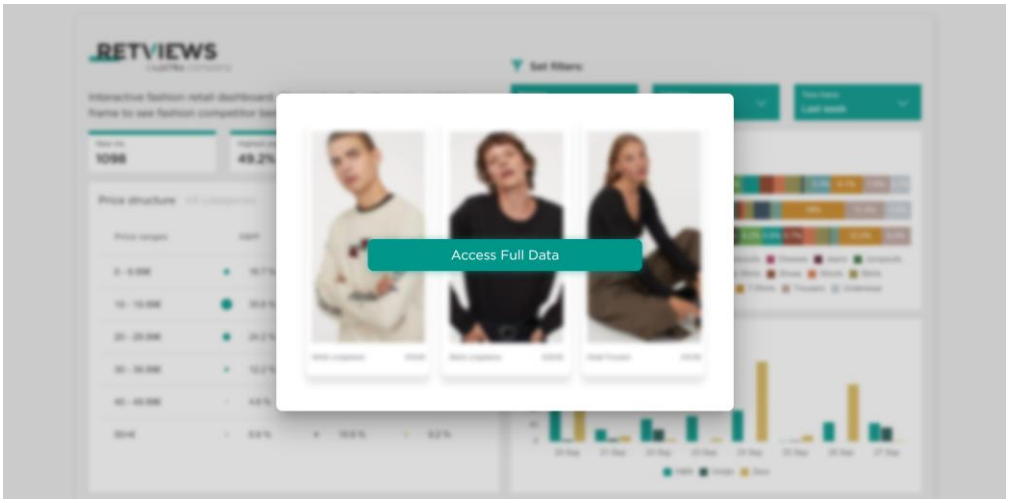

Рис. 4.2. Модальне вікно компонента MarketingDashboardPopoverComponent

LoaderComponent – компонент, відповідальний за відображення індикатора загрузки даних, а саме даних, які будуть надходити до клієнта як JSON об’єкт.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript.

LoaderComponent має наступний вигляд:

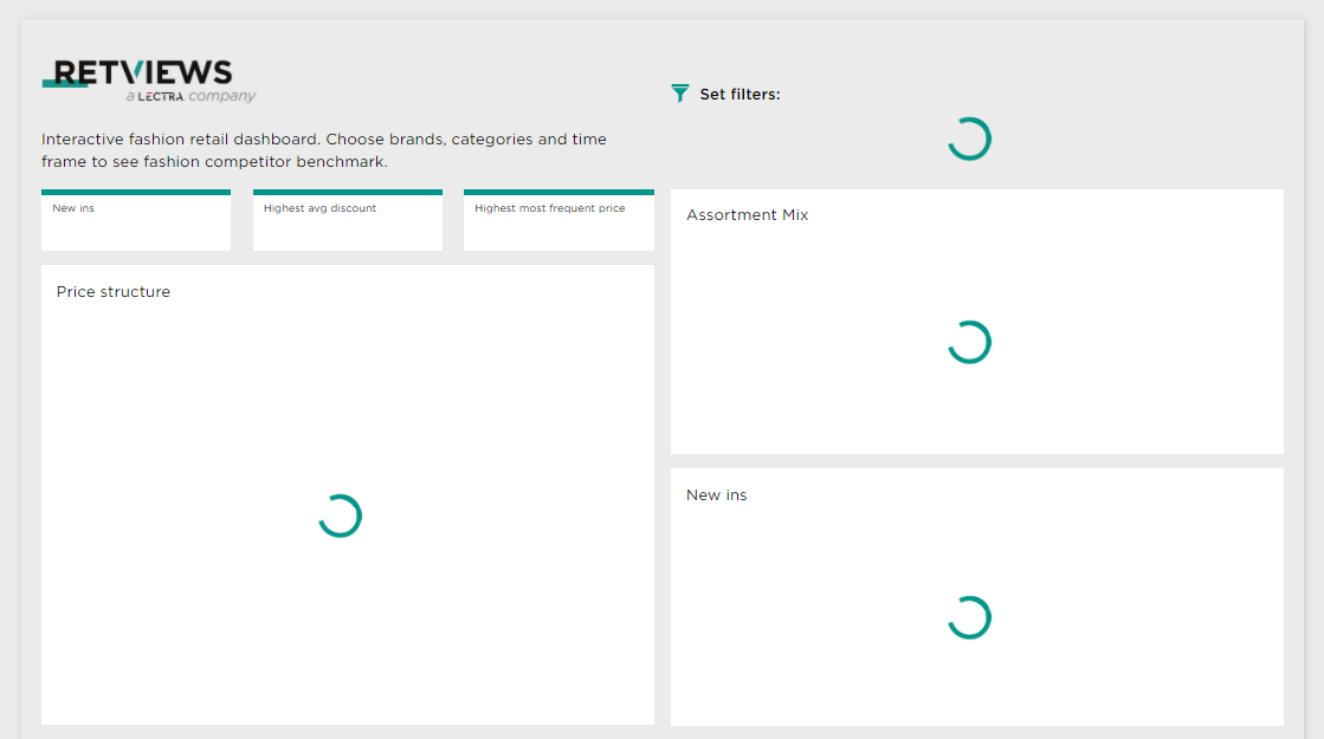

Рис. 4.3. Індикатор загрузки компонента LoaderComponent

MarketingDashboardComponent – найголовніший компонент нашого односторінкового веб-додатку, так як в ньому реазізована логіка зв’язана з фільтрами, графіками та таблицею. В свою чергу цей компонент ієрархічно включає в себе компоненти MarketingDashboardPopoverComponent, LoaderComponent, NoResultsFoundComponent, які будуть динамічно з’являтись на в залежності від користувальницьких дій.

NoResultsFoundComponent – компонент, відповідальний за відображення індикатора відсутності результатів по вибраних фільтрах. Для прикладу в фільтрі «Retailers» нічого не буде вибрано, то даний компонент відобразиться так:

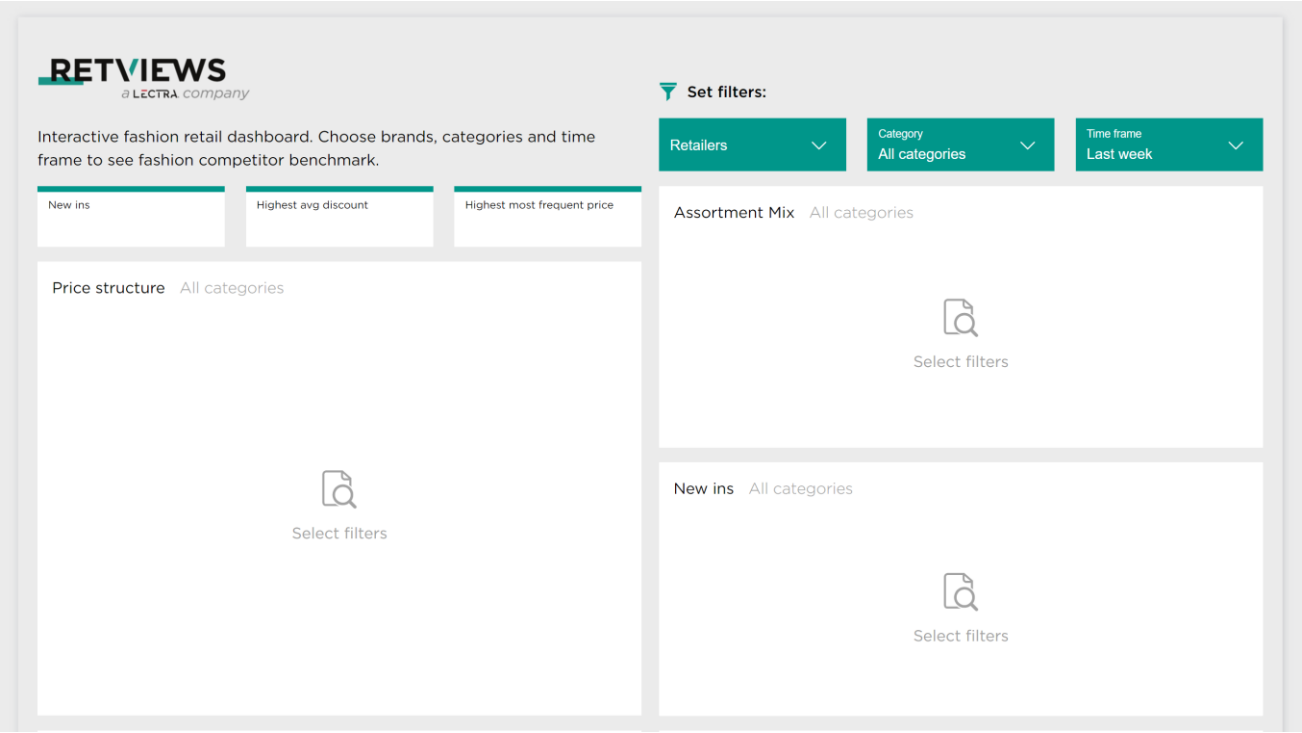

Рис. 4.4. Індикатор відсутності результатів компонента NoResultsFoundComponent

4.7 Реалізація логіки односторінкового веб-додатку

Основна реалізація логіки односторінкового веб-додатку міститься в компоненті MarketingDashboardComponent. Перш за все варто зазначити, як працюють компоненти в Angular фрейморку. Кожен компонент має життєвий цикл. Після створення компонента фреймворк Angular викликає у цього компонента ряд методів, які представляють різні етапи життєвого циклу:

- constructor – викликається, коли Angular створює компонент або директиву, викликаючи «new» (новий екземпляр) для класу.
- ngOnChanges – викликається щоразу, коли відбувається зміна однієї з вхідних властивостей компонента, в тому числі при початковій установці властивостей. Він приймає об'єкт класу SimpleChanges в якості параметра, який містить інформацію про те, які властивості змінились. Можна отримати попередні і поточні значення властивостей.
- ngOnInit – викликається, коли компонент був ініціалізований. Цей метод викликається лише один раз після методу ngOnChanges.

- ngDoCheck – викликається, коли викликається детектор змін компонента. Це дозволяє реалізувати власний алгоритм виявлення змін для даного компонента.
- ngAfterContentInit – викликається один раз після методу ngDoCheck після вставки вмісту в представлення компонента.
- ngAfterContentChecked – викликається кожного разу, коли вміст даного компонента перевіряється механізмом виявлення змін Angular.
- ngAfterViewInit – викликається, коли подання компонента і його дочірніх компонентів повністю ініціалізоване.
- ngAfterViewChecked – викликається фреймворком Angular після перевірки на зміни в поданні компонента.
- ngOnDestroy – буде викликаний безпосередньо перед тим, як Angular знищить компонент. Потрібно використовувати цей метод, щоб скасувати підписку на спостерігаючі об’єкти, щоб уникнути витоків пам'яті.

Рис. 4.5. Життєві цикли Angular компонентів

Для реалізації логіки в MarketingDashboardComponent використовуються два життєві цикли ngOnInit та ngOnDestroy. Життєвий цикл ngOnInit виглядає наступним чином:

```
1 public ngOnInit(): void {
2 this._isLoading = true;
3 this.http.get("get_data.json")
4   .pipe(
5     finalize(() => {
6       this._isLoading = false;
7       this.cdr.detectChanges();
8     })
9   )
10   .subscribe((response: DashboardData) => {
11     this.dashboardData = response;
12     this._filtersForm = this.buildForm();
14     this.init();
15     this.onChangeForm();
16  });
17 }
```

Поле _isLoading – відповідає за відображення LoaderComponent. Бачимо, що властивість _isLoading відповідає значенню «true» означає, що LoaderComponent відобразить індикатор загрузки даних. У наступному рядку 3, відбувається HTTP запит «GET», в якому вказується шлях до веб-сервера. В рядку 10 – підписка на отримання даних від веб-серверу. Як тільки дані будуть отримані, вони будуть присвоєні dashboardData об’єкту. Варто зазначити, що після того, як дані будуть отримані індикатор загрузки зникне, це можна побачити в 6 рядку. 7 рядок відповідає за оновлення компонента MarketingDashboardComponent та його внутрішніх компонентів. В 12 рядку метод класу buildForm() поверне до об’єкту filtersForm дані для фільтрів. В 14 рядку виклакається функція init, яка відповідає за ініціалізацію основних блоків компоненту MarketingDashboardComponent. В 15

рядку виклакається фунція, яка відповідає за відображення змін у випадку, якщо однин з фільтрів зміниться користувачем.
Метод buildForm виглядає натсупним чином:

```
private buildForm(): FormGroup {
  const defaultSelectedBrands = this.dashboardData.brands.filter(brand =>
!brand.locked);
  const defaultSelectedTimeFrame = this._timeFrames.find(timeFrame => timeFrame.id
=== TimeFrames.LastWeek);
  const defaultSelectedCategory = this.dashboardData.categories.find(category =>
category.id === Constants.ALL_CATEGORIES_ID);
  return this.fb.group({
    brands: [defaultSelectedBrands],
    categories: [defaultSelectedCategory],
    timeFrame: [defaultSelectedTimeFrame],
  });
}
```

Можна побачити, що в результаті метод класу повертає об’єкт форми (FormGroup), який буде створений за допомогою конструктора FormBuilder («this.fb.group»). FormBuilder передається у якості сервіса у конструктор. За допомогою методу group створюється об'єкт FormGroup. Кожен елемент передається у форму у звичайному массиві значень. Тут визначено три елементи: «brands», «categories», «timeframe». Для прив'язки об'єкта filtersForm до конкретного елементу форми застосовується атрибут formGroup:

```
<form [formGroup]="_filtersForm">
```

Крім того, необхідно зв'язати об'єкти FormControl (brands, categories, timeframe) з елементами вводу за допомогою атрибутів formControlName:

```
<mat-label>Retailers</mat-label>
<mat-select formControlName="brands" multiple>
```

<mat-label>Category</mat-label>
<mat-select formControlName="categories">
<mat-label>Time frame</mat-label>
<mat-select formControlName="timeFrame">

За допомогою виразів, наприклад «this._filtersForm.controls.timeFrame» ми можемо звернутися до потрібного елементу форми і отримати чи змінити його стан або значення.

4.8 Графіки нових товарів («New ins») та асортиментів («Assortment Mix»)

Базова конфігурація для графіків виглядає наступним чином:

```
public _newInsChart = {
  options: {
    maintainAspectRatio: false,
    responsive: true,
    scales: {
      xAxes: [{
        gridLines: {display: false},
      }],
      yAxes: [{
        gridLines: {display: false},
        ticks: {
          beginAtZero: true,
          callback: (tick, index, array) => (index % 2) ? "" : tick
        }
      }]
    },
    legend: {
```

```
      labels: {boxWidth: 12},
      position: "bottom",
      onHover: onHoverLegend,
    },
    onHover: onHoverChartElements,
    onClick: this.onClickChartElement.bind(this),
  },
  dataSets: null,
  labels: null,
};
```

Ця конфігурація прив’язується до компонента Chart.js, який буде відповідальний за побудову графіку. Поля dataSets та labels будуть заповнені після того, як дані будуть отримані з HTTP запиту. Також варто відмітити, що при зміні одного з фільтрів всі графіки будуть динамічно перемальовані, відповідно даних, які вибрані. Наприклад, виберемо два бренди – H&M і Zara, категорію – T-shirts, період часу – за останній місяць («Last month»).

На першому та на другому графіку (рис. 4.6) відображаються тільки вибрані бренди, категорії та період часу. На другому графіку «New ins» легко проаналізувати нові надходження за весь місяць. Також у випадку зміни фільтра періоду часу на останній тиждень («Last week») користувач побачить дані надходжень за останній тиждень (рис.4.7).

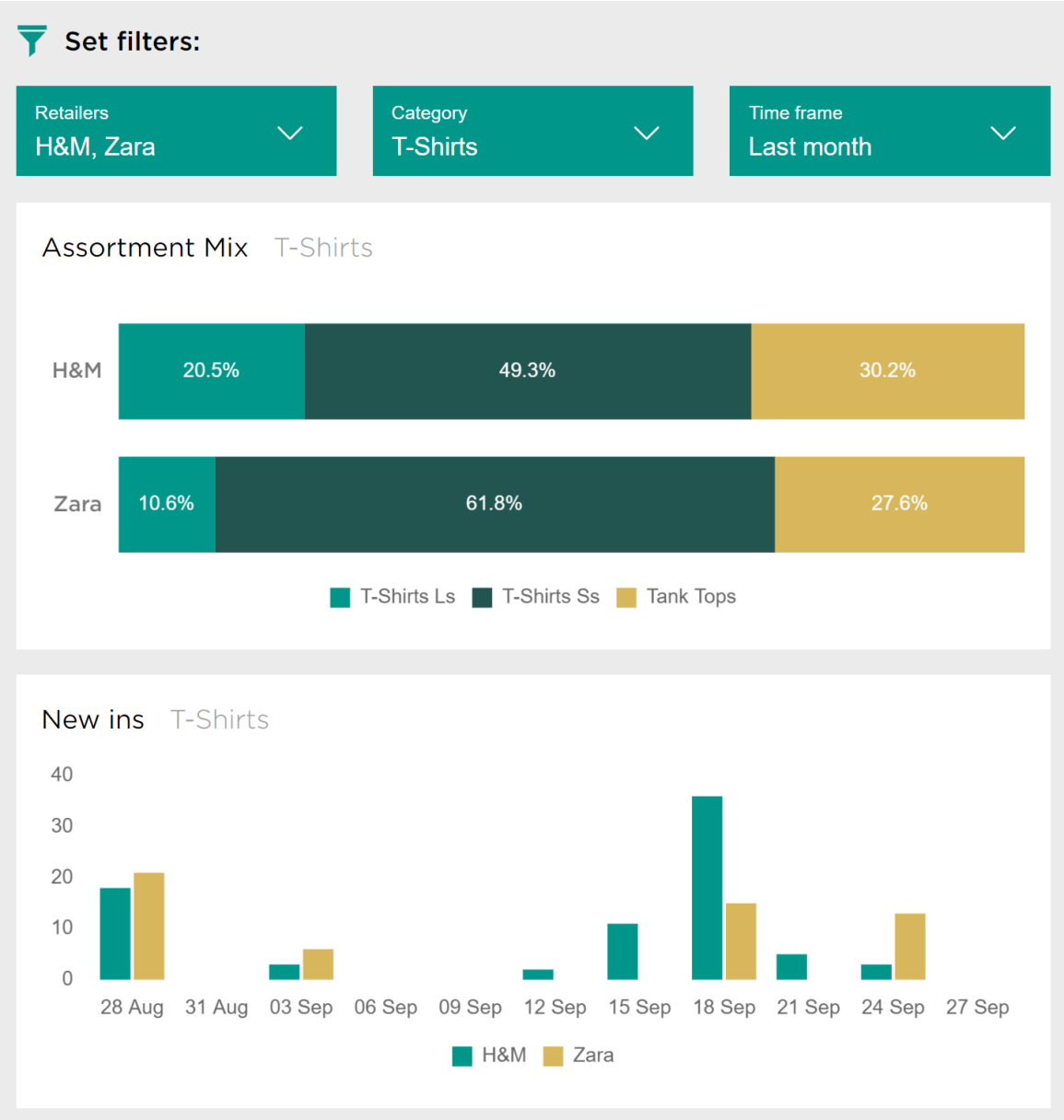

Рис. 4.6. Графіки з налаштованим фільтром

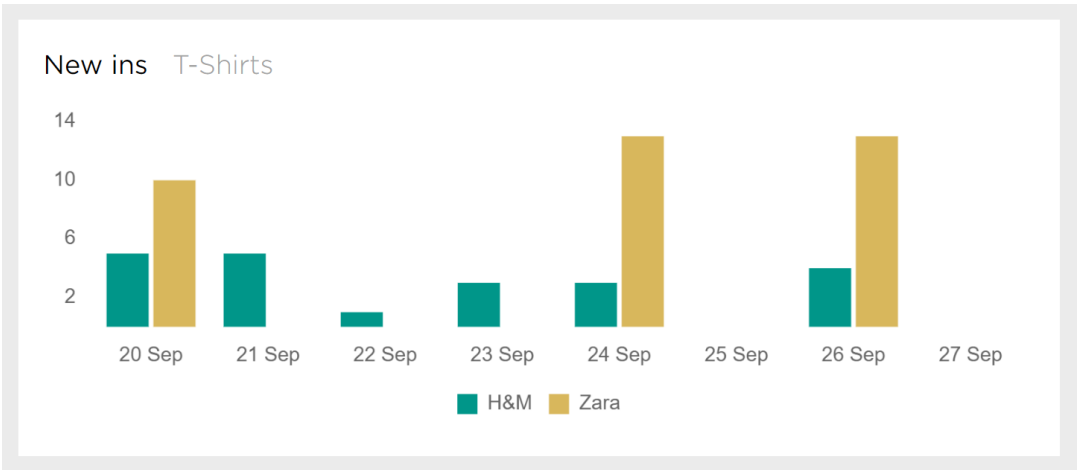

Рис. 4.7. Графік нових товарів «New ins» з періодом часу за останній тиждень

4.9 Таблиця структури цін («Price structure»)

Для побудови таблиці було використано компонент MatTable із бібліотеки Angular Material. Компонент MatTable забезпечує таблицю даних у стилі Material Design, яку можна використовувати для відображення рядків даних. Дана таблиця виглядає наступним чином:

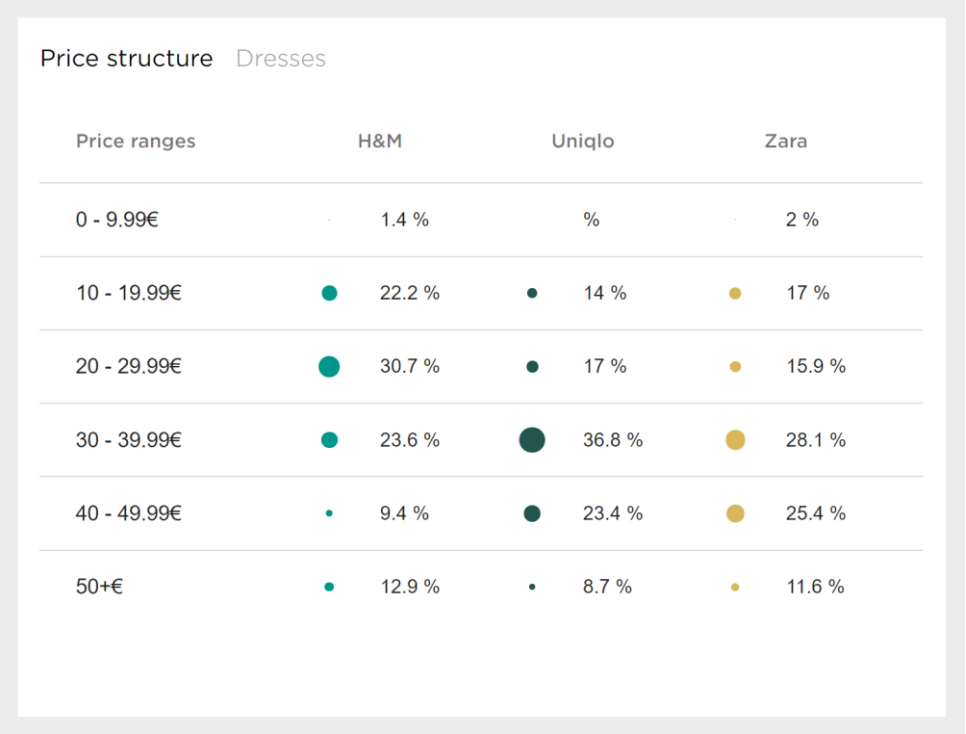

Рис. 4.8. Таблиця структури цін «Price structure»

Таблиця візуалізує дані діапазону цін в залежності від бренду. Щоб користувачу було зручно переглядати дані було прийнято рішення відображати відсотки у вигляді кругів. В такому вигляді інформація легко візуально сприймається. Наприклад, можна зразу побачити найбільший відсоток товарів – 36.8%, в діапазоні цін 30 – 39.99€. Також напроти заголовку таблиці можна побачити вибрану категорію, по якій відображаються дані – у нашому випадку категорія «Dresses».

4.10 Статистичні блоки

Статистичні блоки будуть вираховуватись на основі вибраних фільтрів. Ці блоки будуть представлені у вигляді трьох елементів користувальницького інтерфейсу, які будуть вираховувати:

- Кількість нових товарів («New ins»).
- Найвища знижка («Highest avg discount»).
- Найвища найчастіша ціна («Highest most frequent price»).

Статистичні блоки даних виглядають наступним чином:

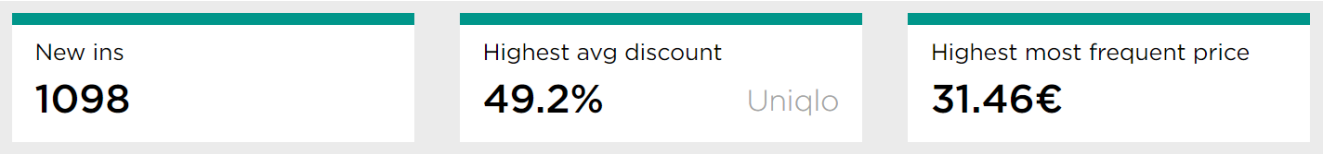

Рис. 4.9. Статистичний блок даних

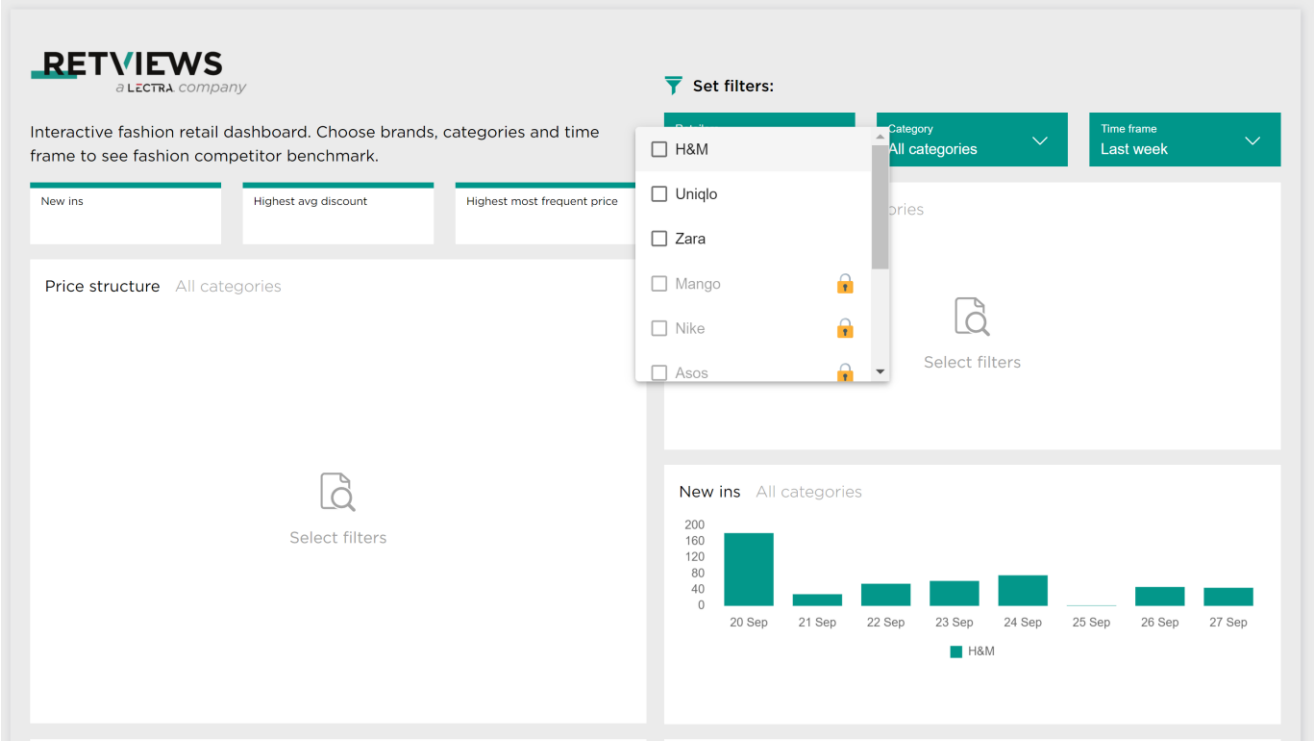

Рис. 4.10. Статистичний блок без вибраног фільтру

Кожен із елементів статистичного блоку, буде вираховуватись на основі вибраних фільтрів – Retailers, Category, Time frame. Кількість нових товарів («New ins») – обчислюється як сума всіх нових товарів. Найвища знижка («Highest avg

discount») – обчислюється як найбільше значення знижки. Напроти значення знижки, можна побачити бренд в якого діє ця знижка. Найвища найчастіша ціна («Highest most frequent price») – обчислюється як найбільша і найчастіше значення. У випадку, якщо не буде вибраного жодного бренду статистиний блок буде виглядати як на рис. 4.10.

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі було описано створення односторінкового веб-додатку. Визначено фунціональні вимоги односторінкового веб-додатку. Підготовлено модель даних для клієнта. Проаналізовано, які вимоги для властивості даних потрібно дотримуватись. Розглянуто основні особливості створення сучасного веб-дизайну та його створення для односторінкового веб-додатку. Підібрано ряд бібліотек для реалізації проекту односторінково веб-додатку панелі управління модної розрібної торгівлі. Розгянуто архітектуру проекту та реалізовано логіку на клієнті для односторінкового веб-додатку. Було створено графіки нових товарів («New ins») та асортиментів («Assortment Mix») таблиця структури цін («Price structure»), статистичні блоки.

ВИСНОВКИ

В ході виконання дипломної роботи було створено односторінковий веб-додаток для ефективного використання в сфері бізнесу за допомогою JavaScript фреймворку Angular 10. Для цього було використано сучасні веб-технології HTML5, CSS3, JavaScript, фреймворк Angular 10.

В результаті створено веб-додаток, який буде корисний для компаній, які займаються бізнес-аналізом світових брендів. Цей додаток являє собою інтерактивну панель для управління модної роздрібної торгівлі сучасних світових брендів, таких як «H&M», «ZARA», «UNIQLO». Ця інтерактивна панель відображає інформацію у вигляді динамічних графіків різного типу та панель управління, за допомогою якої користувач може фільтрувати дані. Створений веб-додаток відповідає принципам сучасного веб-дизайну.

Було проаналізовано основні положення концепції односторінкового веб-додатку та досліджено особливості розробки веб-додатку з використанням Angular 10. Розроблено зручний веб-дизайн для веб-додатку, який повно відображає функціональність.

Матеріали даної дипломної роботи зможуть допомогти залучити більше нових клієнтів, які займаються бізнесом, а саме в бізнесі роздрібної торгівлі. Створений односторінковий веб-додаток може бути використаний, як рекламний продукт такого характеру, що користувачам невідмінно захочеться поділитися інформацією про нього з партнером по бізнесу в сфері продаж.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Документація фреймворку Angular [Електронний ресурс]. – 2020. – режим доступу: https://angular.io (дата звернення 19.10.2020 р).
2. Вибір між традиційними та односторінковими веб-додатками [Електронний ресурс]. – 2020. – режим доступу: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps (дата звернення 27.10.2020 р).
3. Маршрутизація на стороні сервера та клієнта [Електронний ресурс]. – 2019. – режим доступу: https://uk.bccrwp.org/compare/server-side-vs-client-side-routing-cbd027 (дата звернення 05.11.2020 р).
4. Введення в Angular [Електронний ресурс]. – 2020. – режим доступу: https://metanit.com/web/angular2/1.1.php (дата звернення 10.11.2020 р).
5. Модель даних [Електронний ресурс]. – 2020. – режим доступу: https://ru.qaz.wiki/wiki/Data_model (дата звернення 15.11.2020 р).
6. Робота з бібліотекою Chart.js [Електронний ресурс]. – 2017. – режим доступу: https://ruseller.com/lessons.php?rub=32&id=2852 (дата звернення 17.11.2020 р).

ДОДАТОК А

Код програми інтерактивної панелі для управління модної роздрібної торгівлі сучасних світових брендів

HTML компонента marketing-dashboard:

```
<div class="marketing-dashboard">
  <div class="marketing-dashboard__container">
    <div class="marketing-dashboard__row">
      <img class="marketing-dashboard__logo" src="../../../assets/img/logo.png"
alt="Retviews">
    </div>
    <div class="marketing-dashboard__row">
      <div class="marketing-dashboard__column">
        <p class="marketing-dashboard__description">
          Interactive fashion retail dashboard. Choose brands, categories and time frame to
see fashion competitor
          benchmark.</p>
      </div>

      <div class="marketing-dashboard__column">
        <div class="marketing-dashboard__toolbar">
          <div class="marketing-dashboard__toolbar-title">
            <img src="../../../assets/img/icon-filter.svg" alt="icon-filter">
            Set filters:
          </div>

          <form *ngIf="_filtersForm;" [formGroup]="_filtersForm" class="marketing-
dashboard__toolbar-controls"
                novalidate>
            <mat-form-field appearance="fill">
              <mat-label>Retailers</mat-label>
              <mat-select formControlName="brands" multiple>
                <mat-option *ngFor="let brand of dashboardData.brands"
                            [value]="brand"
                            [disabled]="brand.locked"
                            (click)="_onControlOptionClick(brand.locked)"
                            class="marketing-dashboard__toolbar-select">
                  {{brand.value}}
                  <img *ngIf="brand.locked" src="../../../assets/img/locked-icon.svg"
alt="locked-icon"
                       class="locked-icon">
                </mat-option>
              </mat-select>
```

```
        </mat-form-field>
        <mat-form-field appearance="fill">
          <mat-label>Category</mat-label>
          <mat-select formControlName="categories">
            <mat-option class="marketing-dashboard__toolbar-select"
                        *ngFor="let category of dashboardData.categories"
                        [disabled]="category.locked"
                        [value]="category"
                        (click)="_onControlOptionClick(category.locked)">
              {{category.value}}
              <img *ngIf="category.locked" src="../../../assets/img/locked-icon.svg"
alt="locked-icon"
                   class="locked-icon">
            </mat-option>
          </mat-select>
        </mat-form-field>

        <mat-form-field appearance="fill">
          <mat-label>Time frame</mat-label>
          <mat-select formControlName="timeFrame">
            <mat-option class="marketing-dashboard__toolbar-select"
                        *ngFor="let timeFrame of _timeFrames"
                        [disabled]="timeFrame.locked"
                        [value]="timeFrame"
                        (click)="_onControlOptionClick(timeFrame.locked)">
              {{timeFrame.value}}
              <img *ngIf="timeFrame.locked" src="../../../assets/img/locked-icon.svg"
alt="locked-icon"
                   class="locked-icon">
            </mat-option>
          </mat-select>
        </mat-form-field>
      </form>

      <app-loader *ngIf="!_filtersForm && _isLoading"></app-loader>
    </div>
  </div>
</div>

<div class="marketing-dashboard__row">
  <div class="marketing-dashboard__column">
    <div class="marketing-dashboard__statistic">
      <div class="marketing-dashboard__statistic-item">
        <p>New ins</p>
```

```
          <div class="marketing-dashboard__statistic-item-value"
*ngIf="_newInsStatistic && allFiltersSelected">
            {{_newInsStatistic}}
          </div>
        </div>

        <div class="marketing-dashboard__statistic-item">
          <p>Highest avg discount</p>
          <div class="marketing-dashboard__statistic-item-value"
              *ngIf="_highestAvgDiscountStatistic && allFiltersSelected">
            {{_highestAvgDiscountStatistic[1]}}%
<span>{{_highestAvgDiscountStatistic[0]}}</span>
          </div>
        </div>

        <div class="marketing-dashboard__statistic-item">
          <p>Highest most frequent price</p>
          <div class="marketing-dashboard__statistic-item-value"
*ngIf="_highestAvgPriceStatistic && allFiltersSelected">
            {{_highestAvgPriceStatistic[1]}}€</div>
        </div>
      </div>

      <div class="marketing-dashboard__widget marketing-dashboard__widget-price">
        <div class="marketing-dashboard__widget-title">
          Price structure
          <span class="marketing-dashboard__widget-sub-title" *ngIf="_filtersForm">
            {{selectedCategory?.value}}
          </span>
        </div>

        <div class="marketing-dashboard__widget-price-wrapper"
*ngIf="_priceStructure?.data?.length && !_isLoading;">
          <mat-table *ngIf="allFiltersSelected; else selectFiltersTml"
                [dataSource]="_priceStructure.data">
            <ng-container *ngFor="let displayedColumn of
_priceStructure.displayedColumns"
                  [matColumnDef]="displayedColumn">
              <ng-container *ngIf="displayedColumn ===
_constants.PRICE_STRUCTURE">
                <mat-header-cell *matHeaderCellDef>Price ranges</mat-header-cell>
                <mat-cell *matCellDef="let
element">{{element.priceRange}}{{_priceStructure.currency}}</mat-cell>
              </ng-container>
```

```
            <ng-container *ngIf="displayedColumn !==
_constants.PRICE_STRUCTURE">
              <mat-header-cell *matHeaderCellDef>{{displayedColumn}}</mat-header-
cell>
              <mat-cell *matCellDef="let element">
                <div class="percent-cell" (click)="_openPointDetails()">
                  <div class="circle-wrapper">
                    <div class="circle"
                        [style.background-
color]="dashboardData.brandColors[displayedColumn]"
                        [style.width.px]="element[displayedColumn] / 2.15"
                        [style.height.px]="element[displayedColumn] / 2.15">
                    </div>
                  </div>
                  <div class="percent">{{element[displayedColumn] | number:'1.0-1'}}
{{_priceStructure.measurement}} </div>
                </div>
              </mat-cell>
            </ng-container>
          </ng-container>

          <mat-header-row
*matHeaderRowDef="_priceStructure.displayedColumns"></mat-header-row>
          <mat-row *matRowDef="let row; columns:
_priceStructure.displayedColumns;"></mat-row>
        </mat-table>
      </div>

      <app-loader *ngIf="_isLoading"></app-loader>
      <app-no-results-found text="No result found"
                            *ngIf="!_priceStructure?.data && !_isLoading">
      </app-no-results-found>
    </div>
  </div>

  <div class="marketing-dashboard__column">
    <div class="marketing-dashboard__widget marketing-dashboard__widget-
assortment">
      <div class="marketing-dashboard__widget-title">
        Assortment Mix
        <span class="marketing-dashboard__widget-sub-title" *ngIf="_filtersForm">
          {{selectedCategory?.value}}
        </span>
      </div>
```

```
        <div class="marketing-dashboard__widget-assortment-wrapper">
          <ng-container *ngIf="_assortmentMixChart?.dataSets?.length &&
!_isLoading">
            <div *ngIf="allFiltersSelected; else selectFiltersTml"
                class="marketing-dashboard__widget-assortment-wrapper-canvas">
              <canvas baseChart
                    chartType="horizontalBar"
                    [datasets]="_assortmentMixChart.dataSets"
                    [labels]="_assortmentMixChart.labels"
                    [options]="_assortmentMixChart.options"
                    [plugins]="_chartsPlugins"
                    [legend]="true">
              </canvas>
            </div>
          </ng-container>

          <app-loader *ngIf="_isLoading"></app-loader>
          <app-no-results-found text="No result found"
                          *ngIf="!_assortmentMixChart?.dataSets?.length &&
!_isLoading">
          </app-no-results-found>
        </div>
      </div>

      <div class="marketing-dashboard__widget marketing-dashboard__widget-new-
ins">
        <div class="marketing-dashboard__widget-title">
          New ins
          <span class="marketing-dashboard__widget-sub-title" *ngIf="_filtersForm">
            {{selectedCategory?.value}}
          </span>
        </div>

        <ng-container *ngIf="_newInsChart?.dataSets?.length && !_isLoading">
          <div class="marketing-dashboard__widget-new-ins-wrapper-canvas">
            <canvas *ngIf="allFiltersSelected; else selectFiltersTml"
                  baseChart
                  chartType="bar"
                  [datasets]="_newInsChart.dataSets"
                  [labels]="_newInsChart.labels"
                  [options]="_newInsChart.options"
                  [legend]="true">
            </canvas>
          </div>
```

```
        </ng-container>

        <app-loader *ngIf="_isLoading"></app-loader>
        <app-no-results-found text="No result found"
                              *ngIf="!_newInsChart?.dataSets?.length && !_isLoading">
        </app-no-results-found>
       </div>
     </div>
   </div>

   <div class="marketing-dashboard__row">
     <div class="marketing-dashboard__column" *ngFor="let chart of _hiddenCharts">
       <div class="marketing-dashboard__widget">
         <div class="marketing-dashboard__widget-title">
           {{chart.title}}
           <span class="marketing-dashboard__widget-sub-
title">{{chart.subTitle}}</span>
         </div>

         <div class="hidden-chart">
           <img [src]="chart.img" [alt]="chart.title">
           <button class="access-button">Access Full Data</button>
         </div>
       </div>
     </div>
   </div>
  </div>
</div>

<ng-template #selectFiltersTml>
  <app-no-results-found text="Select filters"></app-no-results-found>
</ng-template>
```

TypeScript компонента marketing-dashboard:

```
import {HttpClient} from "@angular/common/http";
import {ChangeDetectionStrategy, ChangeDetectorRef, Component, OnDestroy,
OnInit} from "@angular/core";
import {FormBuilder, FormGroup} from "@angular/forms";
import {MatDialog} from "@angular/material/dialog";
import * as Chart from "chart.js";
import {ChartDataSets} from "chart.js";
import * as pluginDataLabels from "chartjs-plugin-datalabels";
import {Subject} from "rxjs";
```

```
import {finalize, takeUntil} from "rxjs/operators";
import {
  AssortmentMix,
  AvailableCategoriesData,
  DashboardData,
  HiddenCharts,
  PriceStructure,
  SelectItem,
  Statistic,
} from "../../models/dashboard.model";
import {Constants} from "../../utils/constants";
import {MarketingDashboardPopoverComponent} from "../marketing-dashboard-
popover/marketing-dashboard-popover.component";

Chart.defaults.global.plugins.datalabels.display = false;

enum TimeFrames {
  LastWeek = "lastWeek",
  LastMoth = "lastMonth",
  CustomPeriod = "customPeriod"
}

const ASSETS_PATH = "../../../";

function onHoverChartElements(event, elements): void {
  event.target.style.cursor = elements[0] ? "pointer" : "default";
}

function onHoverLegend(event, legendItem): void {
  event.target.style.cursor = legendItem ? "pointer" : "default";
}

@Component({
  selector: "app-marketing-dashboard",
  templateUrl: "./marketing-dashboard.component.html",
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class MarketingDashboardComponent implements OnInit, OnDestroy {
  public dashboardData: DashboardData;
  public _timeFrames: SelectItem[] = [
    {
      id: TimeFrames.LastWeek,
      value: "Last week",
      locked: false
```

```
  },
  {
    id: TimeFrames.LastMoth,
    value: "Last month",
    locked: false
  },
  {
    id: TimeFrames.CustomPeriod,
    value: "Custom period",
    locked: true
  }
];
public _chartsPlugins = [pluginDataLabels];
public _assortmentMixChart = {
  options: {
    scaleShowLabels: false,
    maintainAspectRatio: false,
    responsive: true,
    scales: {
      yAxes: [{
        stacked: true,
        gridLines: {display: false},
        ticks: {
          fontFamily: "Gotham Medium",
          fontSize: 13,
          fontColor: "rgba(0,0,0,.54)"
        }
      }],
      xAxes: [{
        display: false,
        stacked: true,
        gridLines: {display: false},
      }]
    },
    tooltips: {
      mode: "single",
      callbacks: {
        label: (tooltipItem, data) => {
          let label = data.datasets[tooltipItem.datasetIndex].label || "";

          if (label) {
            label += ": ";
          }
          label += tooltipItem.xLabel + "%";
```

```
        return label;
      },
    }
  },
  plugins: {
    datalabels: {
      color: "white",
      display: true,
      clip: true,
      formatter: value => value < 5 ? "" : `${value}%`
    }
  },
  legend: {
    position: "bottom",
    labels: {boxWidth: 12},
    onHover: onHoverLegend,
  },
  onClick: this.onClickChartElement.bind(this),
},
onHover: onHoverChartElements,
dataSets: null,
labels: null,
};
public _newInsChart = {
  options: {
    maintainAspectRatio: false,
    responsive: true,
    scales: {
      xAxes: [{
        gridLines: {display: false},
      }],
      yAxes: [{
        gridLines: {display: false},
        ticks: {
          beginAtZero: true,
          callback: (tick, index, array) => (index % 2) ? "" : tick
        }
      }]
    },
    legend: {
      labels: {boxWidth: 12},
      position: "bottom",
      onHover: onHoverLegend,
    },
```

```
    onHover: onHoverChartElements,
    onClick: this.onClickChartElement.bind(this),
  },
  dataSets: null,
  labels: null,
};
public _highestAvgDiscountStatistic: [string, number];
public _highestAvgPriceStatistic: [string, number];
public _newInsStatistic: number;
public _priceStructure: PriceStructure;
public _constants = Constants;
public _filtersForm: FormGroup;
public _isLoading: boolean;
public _hiddenCharts: HiddenCharts[] = [
  {
    img: `${ASSETS_PATH}assets/img/average-price-hidden.jpg`,
    title: "Average price",
    subTitle: "Average price per category"
  },
  {
    img: `${ASSETS_PATH}assets/img/discount-evolution-hidden.jpg`,
    title: "Discount evolution",
    subTitle: "Discount structure per brand"
  },
  {
    img: `${ASSETS_PATH}assets/img/colors-hidden.jpg`,
    title: "Colors",
    subTitle: "Colors structure per brand"
  },
  {
    img: `${ASSETS_PATH}assets/img/sell-out-hidden.jpg`,
    title: "Sell out",
    subTitle: "Sell out structure per brand"
  },
  {
    img: `${ASSETS_PATH}assets/img/stock-hidden.jpg`,
    title: "Stock",
    subTitle: "Stock structure per brand"
  },
  {
    img: `${ASSETS_PATH}assets/img/fabrics-hidden.jpg`,
    title: "Fabrics",
    subTitle: "Fabrics structure per brand"
  }
```

```
  ];
  private unsubscribe$ = new Subject<void>();

  constructor(private cdr: ChangeDetectorRef,
              private fb: FormBuilder,
              private dialog: MatDialog,
              private http: HttpClient) {
  }

  public get selectedBrands(): SelectItem[] {
    return this._filtersForm.controls.brands.value;
  }

  public get selectedCategory(): SelectItem {
    const categories = this._filtersForm.controls.categories.value;
    return categories ? categories : null;
  }

  public get allFiltersSelected(): boolean {
    return !!(this._filtersForm && this.selectedBrands && this.selectedBrands.length
&& this.selectedCategory && this.selectedTimeFrame);
  }

  public get selectedTimeFrame(): SelectItem {
    const timeFrame = this._filtersForm.controls.timeFrame.value;
    return timeFrame ? timeFrame : null;
  }

  public get availableCategoriesData(): AvailableCategoriesData {
    return this.dashboardData[this.selectedTimeFrame.id][this.selectedCategory.value];
  }

  public ngOnInit(): void {
    this._isLoading = true;
    this.http.get("https://dash.retviews.com/get_data")
      .pipe(
        finalize(() => {
          this._isLoading = false;
          this.cdr.detectChanges();
        })
      )
      .subscribe((response: DashboardData) => {
        this.dashboardData = response;
        this._filtersForm = this.buildForm();
```

```
      this.init();
      this.onChangeForm();
    });
  }

  public ngOnDestroy(): void {
    this.unsubscribe$.unsubscribe();
    this.unsubscribe$.complete();
  }

  public _onControlOptionClick(locked: boolean): void {
    if (locked) {
      window.open("https://google.com");
    }
  }

  public _openPointDetails(): void {
    const dialogRef = this.dialog.open(MarketingDashboardPopoverComponent,
{backdropClass: "overlay-marketing-dashboard-popover"});
    dialogRef.afterClosed().subscribe(result => {
      console.log(`Dialog result: ${result}`);
    });
  }

  private buildForm(): FormGroup {
    const defaultSelectedBrands = this.dashboardData.brands.filter(brand =>
!brand.locked);
    const defaultSelectedTimeFrame = this._timeFrames.find(timeFrame =>
timeFrame.id === TimeFrames.LastWeek);
    const defaultSelectedCategory = this.dashboardData.categories.find(category =>
category.id === Constants.ALL_CATEGORIES_ID);

    return this.fb.group({
      brands: [defaultSelectedBrands],
      categories: [defaultSelectedCategory],
      timeFrame: [defaultSelectedTimeFrame],
    });
  }

  private init(): void {
    if (this.allFiltersSelected) {
      this.initPriceStructureTable();
      this.initNewInsChart();
      this.initAssortmentMixChart();
```

```
    this.initStatisticBoxes();
   }
  }

  private onChangeForm(): void {
   this._filtersForm.valueChanges.pipe(takeUntil(this.unsubscribe$)).subscribe(() =>
this.init());
  }

  private initPriceStructureTable(): void {
   this._priceStructure = this.availableCategoriesData.priceStructure;
   if (this._priceStructure) {
    const priceRangeColumn = this._priceStructure.displayedColumns[0];
    const brandsColumns: string[] = this._filtersForm.controls.brands.value.map((item:
SelectItem) => item.value);
    this._priceStructure.displayedColumns = [priceRangeColumn, ...brandsColumns];
   }
  }

  private initNewInsChart(): void {
   this._newInsChart.dataSets = [];
   this._newInsChart.labels = [];
   const newIns = this.availableCategoriesData.newIns;

   if (newIns) {
    this._newInsChart.dataSets = this.selectedBrands.map((selectedBrand: SelectItem)
=> {
     const brandName = selectedBrand.value;
     const brandData: number[] = newIns.data[brandName];
     const color = this.dashboardData.brandColors[brandName];
     let data: number[];

     switch (this._filtersForm.controls.timeFrame.value.id) {
      case TimeFrames.LastWeek:
       data = brandData;
       this._newInsChart.labels = newIns.timestamps;
       break;
      case TimeFrames.LastMoth:
       data = this.getEveryThirdItem(brandData);
       this._newInsChart.labels = this.getEveryThirdItem(newIns.timestamps);
       break;
     }

     return {
```

```
        label: brandName,
        data: (data && data.length) ? data : [],
        backgroundColor: color,
        hoverBackgroundColor: color,
        hoverBorderColor: color,
        borderColor: color
       } as ChartDataSets;
     });
    }
  }

  private initStatisticBoxes(): void {
    const statistic = this.availableCategoriesData.statistic;
    this._highestAvgPriceStatistic = this.getMaxValueStatistic(statistic.avgPrice);
    this._highestAvgDiscountStatistic = this.getMaxValueStatistic(statistic.avgDiscount);
    this._newInsStatistic = this.getNewInsStatistic(statistic.newIns);
  }

  private getMaxValueStatistic(statisticData: Statistic): [string, number] {
    const selectedBrands = this.selectedBrands.map(item => item.value);
    const filteredStatisticPerBrand = this.filterStatisticPerBrand(statisticData,
selectedBrands);

    const maxValue = Object.entries(filteredStatisticPerBrand)
      .reduce((prev, current) => (prev[1] > current[1]) ? prev : current) as [string,
number];
    return maxValue;
  }

  private getNewInsStatistic(statisticData: Statistic): number {
    const selectedBrands = this.selectedBrands.map(item => item.value);
    const filteredStatisticPerBrand = this.filterStatisticPerBrand(statisticData,
selectedBrands);

    const newIns = Object.values(filteredStatisticPerBrand)
      .reduce((accumulator: number, currentValue: number) => accumulator +
currentValue, 0) as number;
    return newIns;
  }

  private filterStatisticPerBrand(statisticData: Statistic, selectedBrands: string[]):
Statistic {
    return Object.keys(statisticData).reduce((filtered, key) => {
      if (selectedBrands.includes(key)) {
```

```
        filtered[key] = statisticData[key];
      }
      return filtered;
    }, {});
  }

  private initAssortmentMixChart(): void {
    this._assortmentMixChart.dataSets = [];
    this._assortmentMixChart.labels = [];
    const assortmentMix: AssortmentMix = this.availableCategoriesData.assortmentMix;

    if (assortmentMix) {
      const categoriesLabels: string[] = assortmentMix.categoriesLabels;
      const colors = assortmentMix.colors;

      this._assortmentMixChart.dataSets = categoriesLabels.map((category,
categoryIndex) => {
        const data: number[] = this.selectedBrands.map(selectedBrand =>
assortmentMix.data[selectedBrand.value][categoryIndex]);
        const color = colors[categoryIndex];

        return {
          label: category,
          data: (data && data.length) ? data : [],
          backgroundColor: color,
          hoverBackgroundColor: color,
          hoverBorderColor: color,
          borderColor: color
        } as ChartDataSets;
      });
      this._assortmentMixChart.labels = this.selectedBrands.map(item => item.value);
    }
  }

  private getEveryThirdItem<T>(data: Array<T>): Array<T> {
    return data.filter((item, index) => index % 3 === 0);
  }

  private onClickChartElement(event, array): void {
    if (array && array.length) {
      this._openPointDetails();
    }
  }
}
```

SCSS компонента marketing-dashboard:

```
.marketing-dashboard {
  background: #EBEBEB;

  &__container {
    display: flex;
    flex-wrap: wrap;
    max-width: 1300px;
    margin: 20px auto;
    padding: 20px 20px 5px;
    box-shadow: 1px 1px 8px 0 #c5c6cb;
  }

  &__row {
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    width: 100%;
  }

  &__logo {
    width: 220px;
    height: 86px;
  }

  &__description {
    margin-top: 5px;
    margin-bottom: 15px;
  }

  &__column {
    width: 49.3%;
  }

  &__toolbar {
    text-align: center;
    min-height: 108px;

    &-title {
      display: flex;
      align-items: center;
      margin-bottom: 15px;
      margin-top: -41px;
```

```
  font-family: $font-GothamMedium;

  img {
    width: 19px;
    height: 19px;
    margin-right: 10px;
  }
}

&-controls {
  display: flex;
  justify-content: space-between;
}

.mat-form-field {
  width: 31%;
  font-size: 15px;
}

.mat-select {
  &-arrow {
    width: 15px;
    height: 15px;
    background: url('../../img/down-arrow.svg') no-repeat center center;
    border: none;
    color: #fff;
    top: 3px;
    right: 5px;
    position: relative;
  }

  &-value {
    color: #fff;
  }
}

.mat-form-field {
  &-underline {
    display: none;
  }

  &-label {
    color: #fff;
    &-wrapper {
```

```
      top: -1.2em;
    }
  }

  &-appearance-fill {
    .mat-form-field-infix {
      padding: 0;
    }

    .mat-form-field-wrapper {
      padding-bottom: 15px;

      .mat-form-field-flex {
        padding: .85em .75em 0.45em .75em;
        background: #00968A;
        border-radius: 0;
      }
    }
  }

  &.mat-focused .mat-form-field-label {
    color: #fff;
  }
 }
}

&__widget {
 background: #fff;
 padding: 15px;
 margin-bottom: 15px;

 &-title {
  margin-bottom: 15px;
 }

 &-sub-title {
  margin-left: 10px;
  font-family: $font-GothamLight;
  color: #a3a3a3;
 }

 &-price {
  height: 467px;
  display: flex;
```

```
flex-direction: column;

&-wrapper {
  overflow-x: auto;
  flex-grow: 1;
  height: 100%;
}

.mat-table {
  min-width: 450px;
}

.mat-header-cell {
  font-family: $font-GothamMedium;
  justify-content: center;
  font-size: 13px;

  &:first-child {
    justify-content: flex-start;
  }
}

.mat-row {
  &:last-child {
    border: none;
  }
}

.percent-cell {
  display: flex;
  align-items: center;
  width: 100%;
  cursor: pointer;

  .percent {
    width: 50%;
    font-size: 13px;
  }

  .circle {
    border-radius: 50%;

    &-wrapper {
      width: 50%;
```

```
        display: flex;
        justify-content: center;
       }
      }
     }
    }

    &-assortment {
     height: 269px;
     display: flex;
     flex-direction: column;

     &-wrapper {
      height: 100%;
      text-align: center;

      &-canvas {
       height: 200px;
      }
     }
    }

    &-new-ins {
     width: 100%;
     height: 261px;
     display: flex;
     flex-direction: column;

     &-wrapper {
      &-canvas {
       height: 196px;
      }
     }
    }
  }

  &__statistic {
    display: flex;
    justify-content: space-between;
    margin-bottom: 15px;

    &-item {
     width: 31%;
     padding: 5px 11px;
```

```
    border-top: 6px solid #00968A;
    background: #fff;
    min-height: 62.4px;

    p {
      margin: 0;
      font-size: 11px;
    }

    &-value {
      display: flex;
      justify-content: space-between;
      align-items: flex-end;
      font-family: $font-GothamMedium;
      font-size: 19px;

      span {
        margin-left: 10px;
        position: relative;
        top: -3px;
        font-family: $font-GothamLight;
        font-size: 14px;
        color: #a3a3a3;
      }
    }
  }
}


.hidden-chart {
  position: relative;

  img {
    display: block;
    max-width: 100%;
    height: auto;
  }

  .access-button {
    position: absolute;
    left: 50%;
    top: calc(50% - 20px);
    transform: translateX(-50%);
    font-family: $font-GothamLight;
```

```
      text-align: center;
      border: none;
      font-size: 16px;
      padding: 8px 70px;
      background: #00968A;
      color: #fff;
      cursor: pointer;
      outline: none;
      transition: 0.3s ease;
      box-shadow: 1px 1px 8px 3px rgb(255, 255, 255, 0.43);

      &:hover {
        background: #038076;
      }
    }
  }
}

.marketing-dashboard__toolbar-select {
  .mat-pseudo-checkbox-checked {
    background: #00968A;
  }

  .mat-option-text {
    display: flex;
    align-items: center;
    justify-content: space-between;
  }

  &.mat-option {
    &.mat-selected:not(.mat-option-disabled) {
      color: #00968A;
    }

    &[aria-disabled=true] {
      user-select: auto;
      cursor: pointer;
    }
  }

  &:hover {
    background: rgba(0, 0, 0, .04);
  }
  .locked-icon {
```

```
    width: 20px;
    height: 20px;
   }
}

@media (max-width: 1200px) {
  .marketing-dashboard {
    &__container {
      margin-right: 20px;
      margin-left: 20px;
      padding-top: 20px;
    }

    &__row {
      flex-wrap: wrap;
    }

    &__column {
      width: 100%;
    }

    &__toolbar {
      &-title {
        margin-top: 0;
      }
    }

    &__widget {

      &-price {
        height: 425px;
      }

      &-new-ins {
        height: 350px;

        &-wrapper {
          &-canvas {
            height: 285px;
          }
        }
      }
    }
    &__statistic {
```

```
    justify-content: flex-start;

    &-item {
      width: auto;
      min-width: 160px;
      margin-right: 20px;

      &:last-child {
        margin-right: 0;
      }
    }
  }
 }
}

@media (max-width: 768px) {
 .marketing-dashboard {
  &__statistic {
    justify-content: space-between;

    &-item {
      min-width: auto;
      width: 31%;
      margin-right: 0;
    }
  }
 }
}

@media (max-width: 576px) {
 .marketing-dashboard {

  &__statistic {
    flex-wrap: wrap;

    &-item {
      width: 100%;
      margin-right: 0;
      margin-bottom: 20px;
      text-align: center;

      &-value {
        justify-content: center;
      }
```

```
    &:last-child {
      margin-bottom: 0;
    }
  }
}

&__toolbar {
  &-controls {
    flex-direction: column;
  }

  .mat-form-field {
    width: 100%;
  }
}

&__widget {
  &-title {
    display: flex;
    flex-direction: column;
    text-align: center;
  }

  &-price {
    height: 467px;
  }

  &-new-ins {
    height: 375px
  }

  &-assortment {
    height: 350px;

    &-wrapper {
      overflow-x: auto;
      overflow-y: hidden;

      &-canvas {
        width: 450px;
        height: 235px;
      }
    }
  }
```

```
    }
  }
}

@media (max-width: 480px) {
  .marketing-dashboard {
    .hidden-chart {
      img {
        display: none;
      }

      .access-button {
        display: block;
        position: static;
        transform: none;
        margin: 0 auto 5px;
        padding-right: 10px;
        padding-left: 10px;
        width: 100%;
      }
    }
  }
}
```

HTML компонента marketing-dashboard-popover:

```
<mat-dialog-content class="mat-typography">
  <img src="../../../assets/img/popover-hidden.jpg" alt="popover-hidden">
  <button class="access-button">Access Full Data</button>
</mat-dialog-content>
```

TypeScript компонента marketing-dashboard-popover:

```
import {ChangeDetectionStrategy, Component, HostBinding} from "@angular/core";

@Component({
  selector: "app-marketing-dashboard-popover",
  templateUrl: "./marketing-dashboard-popover.component.html",
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class MarketingDashboardPopoverComponent {
  @HostBinding("class") public hostClass = "app-marketing-dashboard-popover";
}
```

SCSS компонента marketing-dashboard-popover:

```
.mat-dialog-container {
  border-radius: 9px !important;
  width: 692px !important;
}

.overlay-marketing-dashboard-popover {
  backdrop-filter: blur(3px);
  background: rgb(90, 90, 90, 0.22);
}

.app-marketing-dashboard-popover {
  .mat-dialog-content {
    position: relative;

    .access-button {
      border-radius: 9px;
      width: 60%;
      position: absolute;
      left: 50%;
      top: calc(50% - 20px);
      transform: translateX(-50%);
      font-family: $font-GothamBook;
      text-align: center;
      border: none;
      font-size: 20px;
      padding: 9px;
      background: #00968A;
      color: #fff;
      cursor: pointer;
      outline: none;
      box-shadow: 1px 1px 8px 3px rgb(0, 150, 138, 0.31);

      &:hover {
        background-color: #038076;
      }
    }

    img {
      display: block;
      height: 430px;
      margin: 0 auto;
    }
  }
}
```

```
@media (max-width: 992px) {
  .app-marketing-dashboard-popover {
    .mat-dialog-content {
      img {
        height: 380px;
      }
    }
  }
}

@media (max-width: 768px) {
  .app-marketing-dashboard-popover {
    .mat-dialog-content {
      img {
        height: 270px;
      }

      .access-button {
        width: 70%;
        font-size: 16px;
      }
    }
  }
}

@media (max-width: 576px) {
  .app-marketing-dashboard-popover {
    .mat-dialog-content {
      img {
        height: auto;
        max-width: 100%;
      }
    }
  }
}

@media (max-width: 480px) {
  .app-marketing-dashboard-popover {
    .mat-dialog-content {
      .access-button {
        padding-right: 0;
        padding-left: 0;
        width: 75%;
      }
```

```
    }
  }
}
```

HTML компонента loader:

```
<div class="lds-ring">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

TypeScript компонента loader:

```
import {ChangeDetectionStrategy, Component, HostBinding} from "@angular/core";

@Component({
  selector: "app-loader",
  templateUrl: "./loader.component.html",
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class LoaderComponent {
  @HostBinding("class") public hostClass = "app-loader";
}
```

SCSS компонента loader:

```
.app-loader {
  text-align: center;
  height: 100%;

  .lds-ring {
    display: inline-block;
    position: relative;
    height: 100%;
  }

  .lds-ring div {
    box-sizing: border-box;
    display: block;
    position: absolute;
    width: 44px;
    height: 44px;
```

```
    margin: 8px;
    border: 5px solid #fff;
    border-radius: 50%;
    animation: lds-ring 1.2s cubic-bezier(0.5, 0, 0.5, 1) infinite;
    border-color: #00968A transparent transparent transparent;
    top: calc(50% - 28px);
    right: calc(50% - 22px);
  }

  .lds-ring div:nth-child(1) {
    animation-delay: -0.45s;
  }

  .lds-ring div:nth-child(2) {
    animation-delay: -0.3s;
  }

  .lds-ring div:nth-child(3) {
    animation-delay: -0.15s;
  }

  @keyframes lds-ring {
    0% {
      transform: rotate(0deg);
    }
    100% {
      transform: rotate(360deg);
    }
  }
}
```

HTML компонента no-results-found:

```
<div class="no-results-found">
  <img src="../../../assets/img/no-results-found.svg" alt="no-results-found">
  <div class="empty-label">{{text}}</div>
</div>
```

TypeScript компонента no-results-found:

```
import {ChangeDetectionStrategy, Component, HostBinding, Input} from
"@angular/core";

@Component({
```

```
  selector: "app-no-results-found",
  templateUrl: "./no-results-found.component.html",
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class NoResultsFoundComponent {
  @Input() public text: string;
  @HostBinding("class") public hostClass = "app-no-results-found";
}
```

SCSS компонента no-results-found:

```
.app-no-results-found {
  height: 100%;

  .no-results-found {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    flex-grow: 1;
    color: #a3a3a3;
    height: 100%;

    .empty-label {
      padding-top: 13px;
    }
  }
}
```

Файл base.scss:

```
@import "vars";
@import "fonts";

* {
  box-sizing: border-box;
  line-height: 1.5;
}

body {
  margin: 0;
  background: #EBEBEB;
  font-family: $font-GothamBook;
}
```

```
@import "components/marketing-dashboard";
@import "components/loader";
@import "components/no-results-found";
@import "components/marketing-dashboard-popover";
```

Файл fonts.scss:

```
@import "mixins";
@include font-face("Gotham Book", "../fonts/Gotham-Book", 400);
@include font-face("Gotham Medium", "../fonts/Gotham-Medium", 500);
@include font-face("Gotham Light", "../fonts/Gotham-Light", 300);
```

Файл mixins.scss:

```
@mixin font-face($font-family, $file-path, $weight: normal, $style: normal, $asset-pipeline: false ) {
  @font-face {
    font-family: $font-family;
    font-weight: $weight;
    font-style: $style;

    @if $asset-pipeline == true {
      src: font-url('#{$file-path}.eot');
      src: font-url('#{$file-path}.eot?#iefix') format('embedded-opentype'), font-url('#{$file-path}.woff') format('woff'), font-url('#{$file-path}.ttf') format('truetype');
    } @else {
      src: url('#{$file-path}.eot');
      src: url('#{$file-path}.eot?#iefix') format('embedded-opentype'), url('#{$file-path}.woff') format('woff'), url('#{$file-path}.ttf') format('truetype');
    }
  }
}
```

Файл vars.scss:

```
$font-GothamBook: "Gotham Book", sans-serif;
$font-GothamMedium: "Gotham Medium", sans-serif;
$font-GothamLight: "Gotham Light", sans-serif;
```

Інерфейс dashboard.model:

```
export interface DashboardData {
  brands: SelectItem[];
  brandColors: { [key: string]: string };
```

```
  categories: SelectItem[];
  lastWeek: AvailableCategories;
  lastMonth: AvailableCategories;
}

export interface AvailableCategories {
  [key: string]: AvailableCategoriesData;
}

export interface AvailableCategoriesData {
  assortmentMix: AssortmentMix;
  priceStructure: PriceStructure;
  newIns: NewIns;
  statistic: {
    avgDiscount: Statistic;
    avgPrice: Statistic;
    newIns: Statistic;
  };
}

export interface Statistic {
  [key: string]: number;
}

export interface SelectItem {
  id?: string;
  value: string;
  locked?: boolean;
}

export interface PriceStructure {
  data: PriceStructureData[];
  displayedColumns: string[];
  measurement: string;
  currency: string;
}

interface PriceStructureData {
  [key: string]: string | number;
}

interface NewIns {
  data: { [key: string]: number[] };
  timestamps: string[];
```

```
}

export interface AssortmentMix {
  categoriesLabels: string[];
  data: { [key: string]: number[] };
  colors: string[];
}

export interface HiddenCharts {
  img: string;
  title: string;
  subTitle: string;
}
```

Допоміжні константи:

```
export class Constants {
  public static readonly ALL_CATEGORIES_ID: string = "all_categories";
  public static readonly PRICE_STRUCTURE: string  = "Price structure";
}
```

HTML компонента app:

```
<app-marketing-dashboard></app-marketing-dashboard>
```

TypeScript компонента app:

```
import {Component} from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
})
export class AppComponent {
}
```

Модуль app:

```
import {HttpClientModule} from "@angular/common/http";
import {NgModule} from "@angular/core";
import {FormsModule, ReactiveFormsModule} from "@angular/forms";
import {MatButtonModule} from "@angular/material/button";
import {MatCheckboxModule} from "@angular/material/checkbox";
import {MatOptionModule} from "@angular/material/core";
```

```
import {MatDialogModule} from "@angular/material/dialog";
import {MatFormFieldModule} from "@angular/material/form-field";
import {MatSelectModule} from "@angular/material/select";
import {MatTableModule} from "@angular/material/table";
import {BrowserModule} from "@angular/platform-browser";
import {BrowserAnimationsModule} from "@angular/platform-browser/animations";
import {ChartsModule} from "ng2-charts";
import {AppComponent} from "./app.component";
import {LoaderComponent} from "./components/loader/loader.component";
import {MarketingDashboardPopoverComponent} from "./components/marketing-
dashboard-popover/marketing-dashboard-popover.component";
import {MarketingDashboardComponent} from "./components/marketing-
dashboard/marketing-dashboard.component";
import {NoResultsFoundComponent} from "./components/no-results-found/no-results-
found.component";

@NgModule({
  declarations: [
    AppComponent,
    MarketingDashboardComponent,
    MarketingDashboardPopoverComponent,
    LoaderComponent,
    NoResultsFoundComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
    ChartsModule,
    MatFormFieldModule,
    MatOptionModule,
    MatSelectModule,
    BrowserAnimationsModule,
    MatCheckboxModule,
    MatTableModule,
    MatDialogModule,
    MatButtonModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {
}
```

NPM пакети:

```
{
  "name": "project",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~10.1.0-next.4",
    "@angular/cdk": "^10.1.2",
    "@angular/common": "~10.1.0-next.4",
    "@angular/compiler": "~10.1.0-next.4",
    "@angular/core": "~10.1.0-next.4",
    "@angular/forms": "~10.1.0-next.4",
    "@angular/material": "^10.1.2",
    "@angular/platform-browser": "~10.1.0-next.4",
    "@angular/platform-browser-dynamic": "~10.1.0-next.4",
    "@angular/router": "~10.1.0-next.4",
    "chart.js": "^2.9.3",
    "chartjs-plugin-datalabels": "^0.7.0",
    "ng2-charts": "^2.3.2",
    "rxjs": "~6.6.0",
    "tslib": "^2.0.0",
    "zone.js": "~0.10.2"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.1001.0-next.4",
    "@angular/cli": "~10.1.0-next.4",
    "@angular/compiler-cli": "~10.1.0-next.4",
    "@types/node": "^12.11.1",
    "@types/jasmine": "~3.5.0",
    "@types/jasminewd2": "~2.0.3",
    "codelyzer": "^6.0.0",
    "jasmine-core": "~3.6.0",
    "jasmine-spec-reporter": "~5.0.0",
    "karma": "~5.0.0",
    "karma-chrome-launcher": "~3.1.0",
```

```
    "karma-coverage-istanbul-reporter": "~3.0.2",
    "karma-jasmine": "~3.3.0",
    "karma-jasmine-html-reporter": "^1.5.0",
    "protractor": "~7.0.0",
    "ts-node": "~8.3.0",
    "tslint": "~6.1.0",
    "typescript": "~3.9.5"
  }
}
```

Файл tsconfig.app.json:

```
{
  "extends": "./tsconfig.base.json",
  "compilerOptions": {
    "outDir": "./out-tsc/app",
    "types": []
  },
  "files": [
    "src/main.ts",
    "src/polyfills.ts"
  ],
  "include": [
    "src/**/*.d.ts"
  ]
}
```

Файл tsconfig.base.json:

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "importHelpers": true,
    "target": "es2015",
    "module": "es2020",
    "lib": [
      "es2018",
```

```
      "dom"
    ]
  }
}
```

Файл tsconfig.json:

```
{
  "files": [],
  "references": [
    {
      "path": "./tsconfig.app.json"
    },
    {
      "path": "./tsconfig.spec.json"
    }
]
}
```

Файл tsconfig.spec.json:

```
{
  "extends": "./tsconfig.base.json",
  "compilerOptions": {
    "outDir": "./out-tsc/spec",
    "types": [
      "jasmine"
    ]
  },
  "files": [
    "src/test.ts",
    "src/polyfills.ts"
  ],
  "include": [
    "src/**/*.spec.ts",
    "src/**/*.d.ts"
  ]
}
```

Файл angular.json:

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
```

```
"newProjectRoot": "projects",
"projects": {
 "test-ap": {
  "projectType": "application",
  "schematics": {
   "@schematics/angular:component": {
    "style": "scss"
   }
  },
  "root": "",
  "sourceRoot": "src",
  "prefix": "app",
  "architect": {
   "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {
     "outputPath": "dist/test-ap",
     "index": "src/index.html",
     "main": "src/main.ts",
     "polyfills": "src/polyfills.ts",
     "tsConfig": "tsconfig.app.json",
     "aot": true,
     "assets": [
      "src/favicon.ico",
      "src/assets"
     ],
     "styles": [
      "./node_modules/@angular/material/prebuilt-themes/indigo-pink.css",
      "src/styles.scss"
     ],
     "scripts": []
    },
    "configurations": {
     "production": {
      "fileReplacements": [
       {
        "replace": "src/environments/environment.ts",
        "with": "src/environments/environment.prod.ts"
       }
      ],
      "optimization": true,
      "outputHashing": "all",
      "sourceMap": false,
      "extractCss": true,
```

```
      "namedChunks": false,
      "extractLicenses": true,
      "vendorChunk": false,
      "buildOptimizer": true,
      "budgets": [
        {
          "type": "initial",
          "maximumWarning": "2mb",
          "maximumError": "5mb"
        },
        {
          "type": "anyComponentStyle",
          "maximumWarning": "6kb",
          "maximumError": "10kb"
        }
      ]
    }
  }
},
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  "options": {
    "browserTarget": "test-ap:build"
  },
  "configurations": {
    "production": {
      "browserTarget": "test-ap:build:production"
    }
  }
},
"extract-i18n": {
  "builder": "@angular-devkit/build-angular:extract-i18n",
  "options": {
    "browserTarget": "test-ap:build"
  }
},
"test": {
  "builder": "@angular-devkit/build-angular:karma",
  "options": {
    "main": "src/test.ts",
    "polyfills": "src/polyfills.ts",
    "tsConfig": "tsconfig.spec.json",
    "karmaConfig": "karma.conf.js",
    "assets": [
```

```
            "src/favicon.ico",
            "src/assets"
          ],
          "styles": [
            "./node_modules/@angular/material/prebuilt-themes/indigo-pink.css",
            "src/styles.scss"
          ],
          "scripts": []
        }
      },
      "lint": {
        "builder": "@angular-devkit/build-angular:tslint",
        "options": {
          "tsConfig": [
            "tsconfig.app.json",
            "tsconfig.spec.json",
            "e2e/tsconfig.json"
          ],
          "exclude": [
            "**/node_modules/**"
          ]
        }
      },
      "e2e": {
        "builder": "@angular-devkit/build-angular:protractor",
        "options": {
          "protractorConfig": "e2e/protractor.conf.js",
          "devServerTarget": "test-ap:serve"
        },
        "configurations": {
          "production": {
            "devServerTarget": "test-ap:serve:production"
          }
        }
      }
    }
  },
  "defaultProject": "test-ap"
}
```